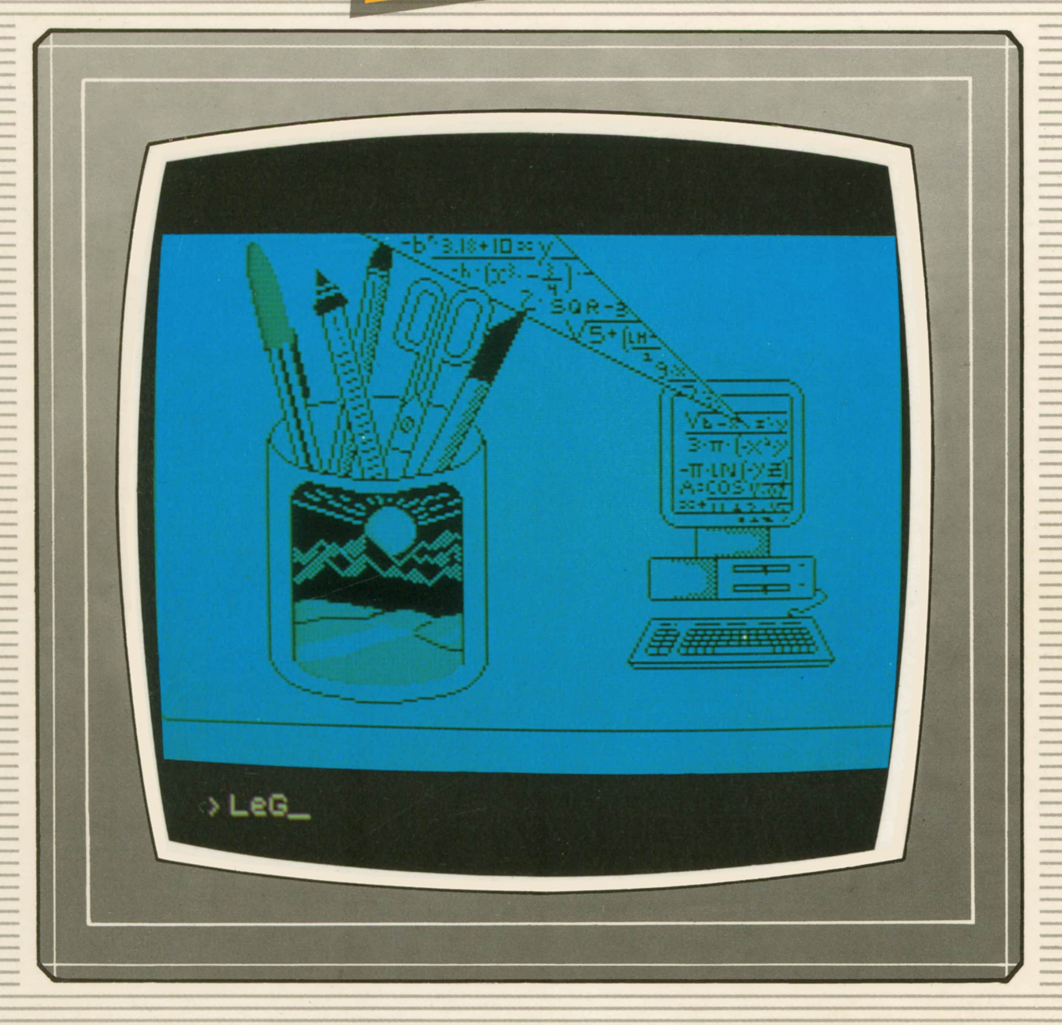
▼ AMSTRAD ▼ SPECTRUM ▼ COMMODORE ▼ IBM ▼ MSX ▼

Informática y programación 28

PASO A PASO



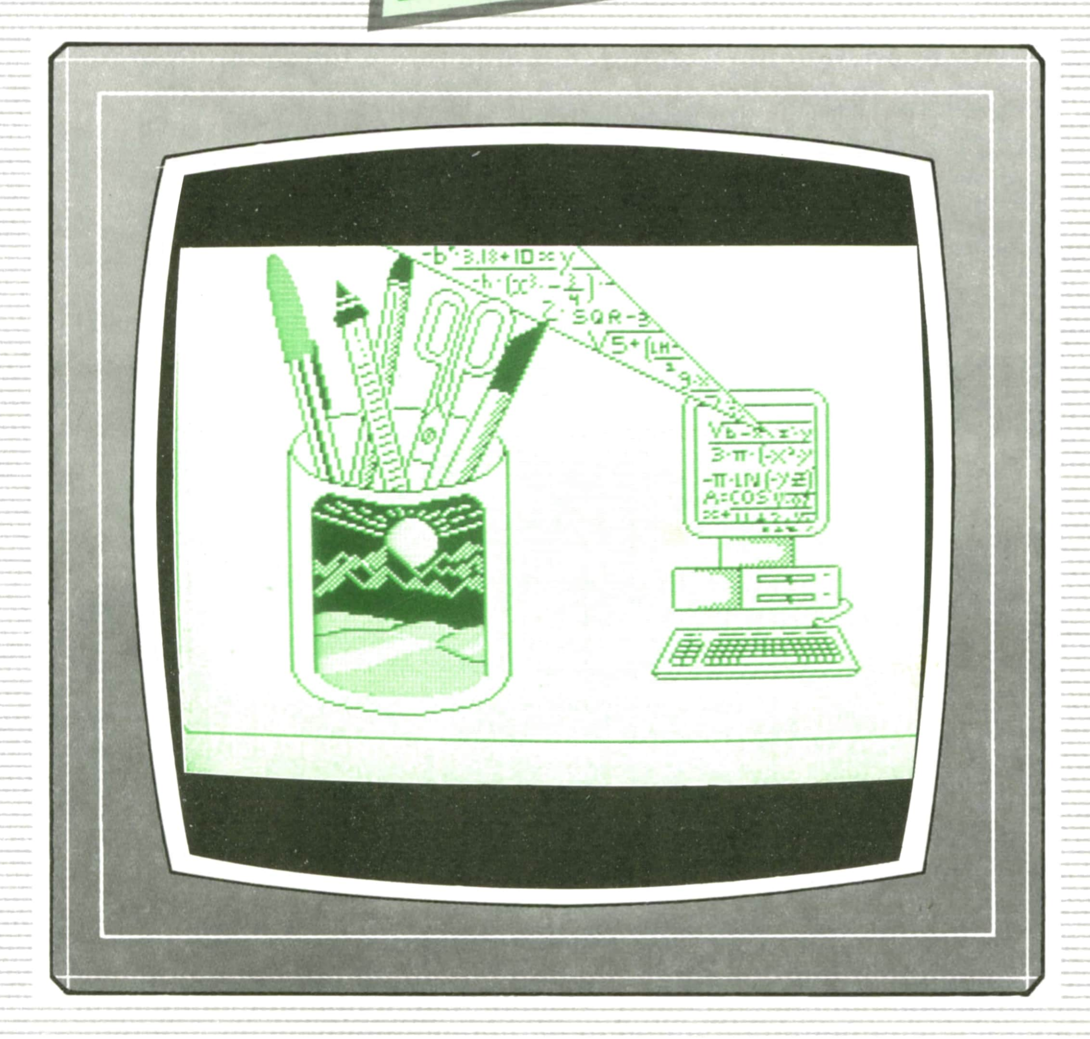
PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Informática y programación 28

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Una publicación de

EDICIONES SIGLO CULTURAL, S.A.

Director-editor:
RICARDO ESPAÑOL CRESPO.

Gerente:
ANTONIO G. CUERPO.

Directora de producción:
MARIA LUISA SUAREZ PEREZ.

Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación
y Licenciado en Informática.
JOSE ARTECHE, Ingeniero de Telecomunicación.

Diseño y maquetación:
BRAVO-LOFISH.

Fotografía:
EQUIPO GALATA.

Dibujos:
JOSE OCHOA

TECNICAS DE PROGRAMACION: Manuel Alfonseca, Doctor Ingeniero de Telecomunicación y Licenciado en Informática. TECNICAS DE ANALISIS: José Arteche, Ingeniero en Telecomunicación. LENGUAJE MAQUINA 8086: Juan Rojas Licenciado en Ciencias Físicas e Ingeniero Industrial. PASCAL: Juan Ignacio Puyol, Ingeniero Industrial. PROGRAMAS (educativos, de utilidad, de gestión y de juegos): Francisco Morales, Técnico en Informática y colaboradores. Coordinador de AULA DE INFORMATICA APLICADA (AIA): Alejandro Marcos, Licenciado en Ciencias Químicas. BASIC: Esther Maldonado, Diplomada en Arquitectura. INFORMATICA BASICA: Virginia Muñoz, Diplomada en Informática. LENGUAJE MAQUINA Z-80: Joaquín Salvachúa, Diplomado en Telecomunicación y José Luis Tojo, Diplomado en Telecomunicación. LENGUAJE MAQUINA 6502: (desde el tomo 5): Juan José Gómez, Licenciado en Química. LOGO: Cristina Manzanera, Licenciada en Informática. APLICACIONES: Jorge Thomas, Técnico de Informática. OTROS LENGUAJES (COBOL): Joaquín Salvachúa, Diplomado en Telecomunicación. José Luis Tojo, Diplomado en Telecomunicación.

Ediciones Siglo Cultural, S.A.

Dirección, redacción y administración:
Pedro Teixeira, 8, 2.ª planta. Teléf. 455 09 99. 28020 Madrid.

Publicidad:
Gofar Publicidad, S.A. Benito de Castro, 12 bis. 28028 Madrid.

Distribución en España:
COEDIS, S.A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.

Distribución en Ecuador: Muñoz Hnos.

Distribución en Perú: DISELPESA.

Distribución en Chile: Alfa Ltda.

Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América, 1532. Teléf.: 21 24 64.
Buenos Aires - 1.290. Argentina.

ISBN del tomo: 84-7688-168-1
ISBN de la obra: 84-7688-068-7

Fotocomposición:
ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.

Imprime:
MATEU CROMO. Pinto (Madrid).

Depósito legal: M-5-677-1987
Printed in Spain - Impreso en España.

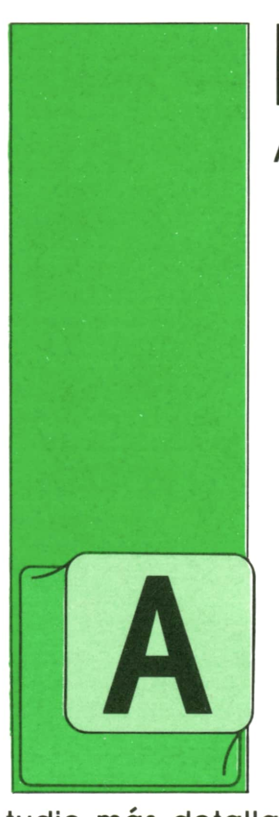
Suscripciones y números atrasados:
Ediciones Siglo Cultural, S.A.
Pedro Teixeira, 8, 2.ª planta. Teléf. 259 73 31. 28020 Madrid.

Diciembre, 1987.
P.V.P. Canarias: 335,–.

INDICE

INFORMATICA BASICA

ANALISIS DE SISTEMAS



UNQUE en un capítulo anterior vimos el análisis de sistemas como parte integrante del proceso de mecanización, vamos a profundizar más en el tema haciendo un estudio más detallado del análisis funcional y el orgánico.



Análisis funcional

Esta fase de análisis comienza una vez finalizada la de estudio de viabilidad del proyecto informático. Se desarrolla en líneas generales informáticas, no en programas concretos. Su objetivo es diseñar un nuevo sistema.

Primeramente hay que tener un profundo conocimiento de las necesidades del

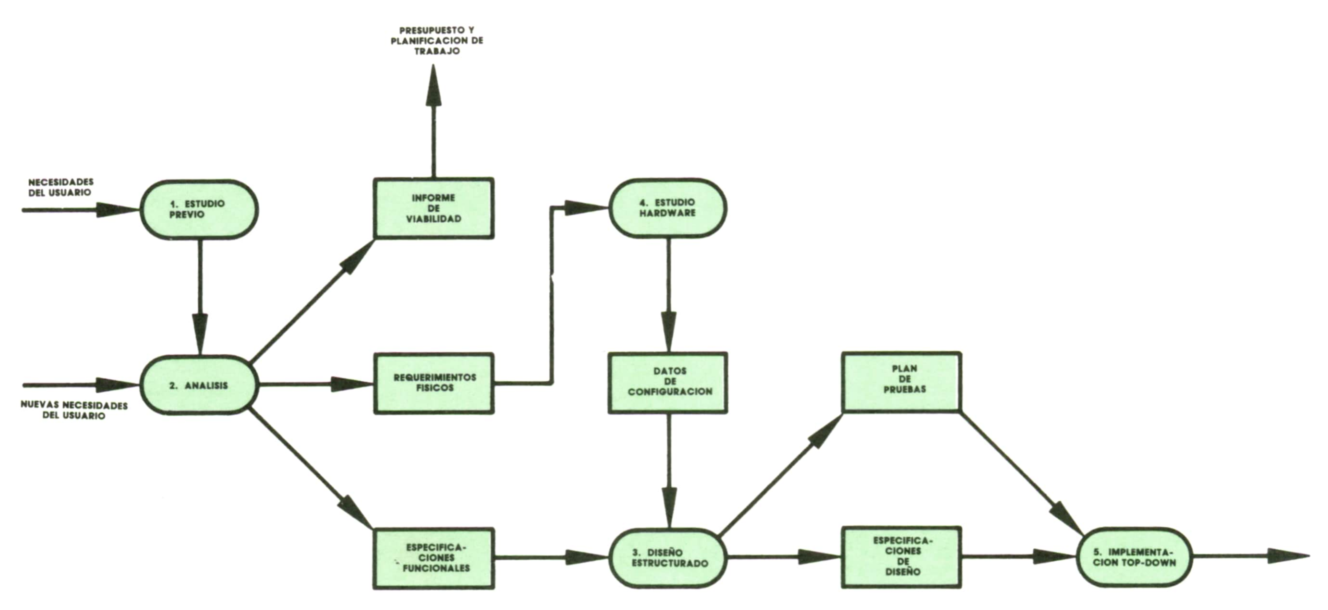



Fig. 1.

sistema actual para conocer detalladamente las áreas que puedan ser objeto de mejoras y los problemas existentes. Para ello es necesario hacer una descripción general del sistema en cuanto a ámbito, normas e instrucciones existentes.

Este estudio se hace a distintos niveles:

— estudio lógico de cada documento existente,

— estudio físico del soporte de información,

— evaluación global del sistema actual, fijándonos en los siguientes puntos:

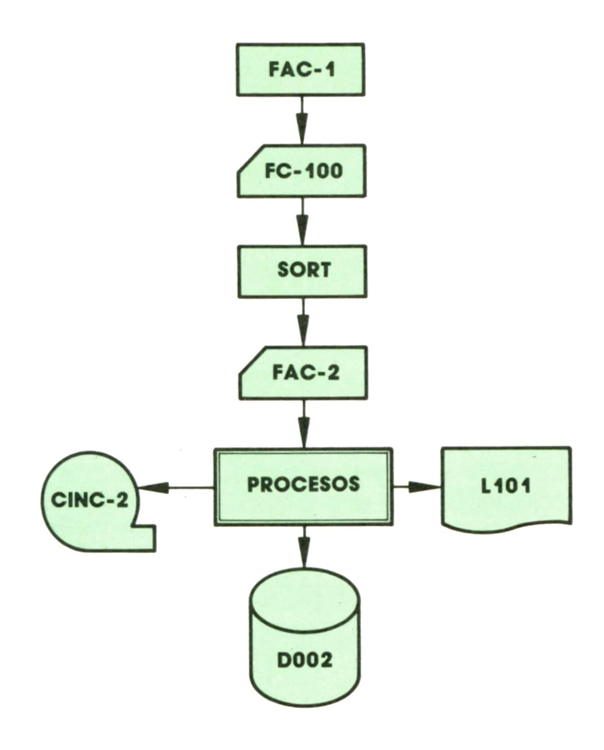
— funciones innecesarias,

— documentación deficiente,
— nuevas necesidades,
— falta o deficiencia de controles, etc.

Una vez conocidas las necesidades de información se debe diseñar totalmente el nuevo método de trabajo, definiendo qué datos se suministrarán desde el exterior, cuáles van a ser los procedimientos utilizados y las nuevas salidas para producir los informes.

También se determinarán los procedimientos de actualización y agrupamiento de datos y, por último, se establecerá la sucesión de los distintos tratamientos necesarios.

Aunque en el momento de la mecanización no se haya pensado más que en una parte del trabajo, el análisis debe realizarse sobre todo él. Si lo que se va a automatizar es una empresa, probablemente existan departamentos en los que ya se haya analizado una aplicación similar o incluso idéntica.




Ejemplo de un organigrama funcional.

Fases del análisis funcional

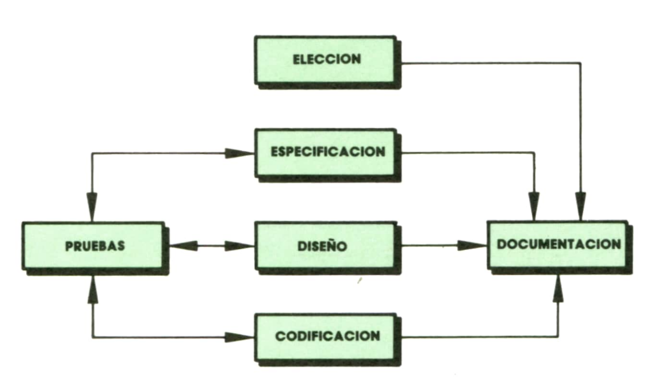
Los puntos principales a desarrollar en un análisis funcional son tres: informes, entidades y procesos. Las fases del análisis funcional se corresponden con ellos, excepto una fase adicional de tratamiento global.

1. *Esquema general*

Inicialmente se hace un análisis global y se determinan las divisiones del trabajo en distintas áreas. Se intenta hacer una descripción general de la aplicación desde el punto de vista organizativo, en forma jerárquica. Si algunas de las áreas de subdivisión sigue siendo muy compleja, se vuelve a efectuar una división dentro de ella. Con este análisis descendente se va produciendo un árbol invertido, en cuyos últimos nodos se encontrarán las informaciones necesarias para tratar cada una de las áreas.

Cada una de estas áreas tendrá un tratamiento diferente y los procesos correspondientes serán independientes, con distintos procedimientos y operaciones. Será necesario, por tanto, prever las necesidades de cada proceso (hardware, software, personal, etc.) y diseñar los nuevos documentos.

Una vez definidas completamente cada una de estas áreas, se hará de nuevo el análisis ascendente, estableciendo las relaciones con el resto de aplicaciones.




Fases en un proyecto informático.

2. *Detalle de informes*

En cada uno de los informes obtenidos del análisis anterior se hará hincapié en dos aspectos; uno en cuanto al contenido, poniendo especial interés en los formatos y otro en cuanto a las líneas de información.

El dossier del análisis funcional tiene

por objetivo describir detalladamente todas las acciones llevadas a cabo para obtener la solución adecuada; para ello se redacta un informe sobre las necesidades y peticiones del usuario. Este informe se llama *cuaderno de carga* y puede ser diseñado por el informático, el cliente o el usuario.

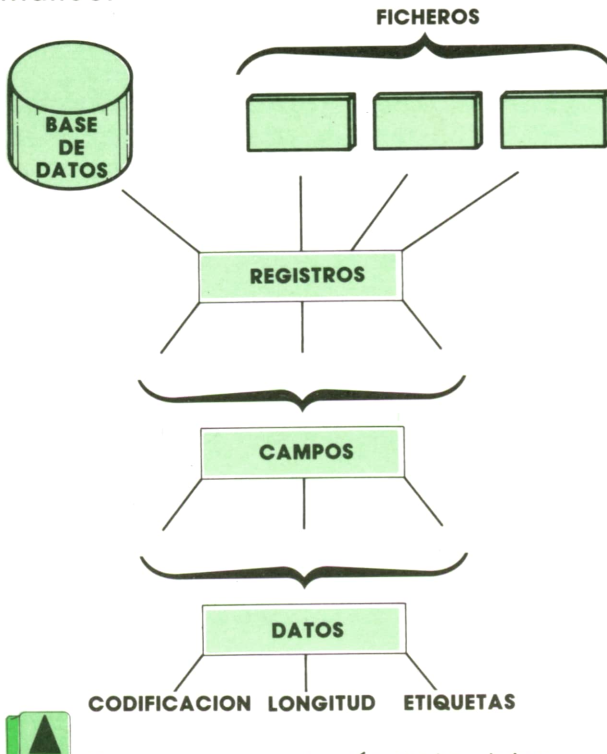
Los puntos básicos en el diseño del cuaderno de carga son:

— Definición general del problema: su naturaleza, su periodicidad en las explotaciones de los datos, relaciones del sistema con el resto de las áreas de la empresa, etc.

— Los documentos base.

— Los resultados.

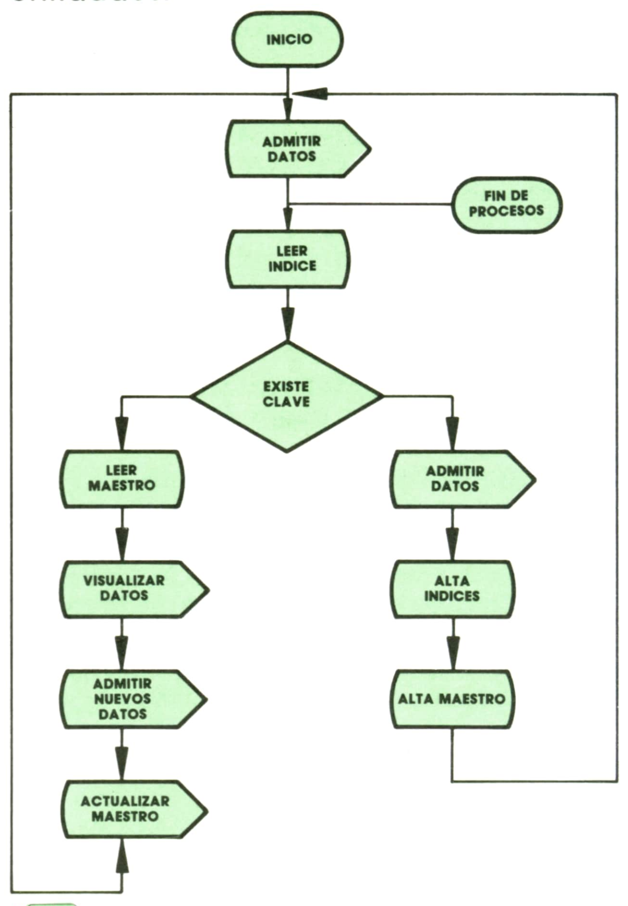
— El tratamiento: esta parte deberá elaborarse con mucho cuidado y muy claramente para una perfecta y rápida comprensión por parte del personal informático.



Proceso de descripción de los datos.

3. *Estructura de la información*

Consiste en organizar las informaciones y datos del punto anterior, haciendo una subdivisión basada en las áreas en las que se aplica. Una vez hecho el reparto, se especifican sus características referentes, por ejemplo, al tipo de dato, número de posiciones necesarias, etc. También hay que tener en cuenta que no todos los datos se tienen que agrupar en entidades.





Organigrama de bloques.

4. *Análisis de procesos*

Consiste en hacer un estudio en profundidad de los procesos a realizar en cada una de las áreas de subdivisión del primer punto, para mantener los datos actualizados. Para cada uno se realizará un diagrama con sus entradas y salidas y la descripción de sus objetivos.



Análisis orgánico

Una vez analizada funcionalmente la aplicación se deberá facilitar toda la información necesaria y suficiente para la programación y puesta a punto, convirtiendo el diseño funcional en un diseño detallado, desde el punto de vista de sus características informáticas.

Desde una visión global se deberán tratar tanto la estructura general de la aplicación en cuanto a configuración hardware y software, como la división de la aplicación en procesos, determinan-

do las relaciones concretas entre las aplicaciones.

Las normas de programación y puesta a punto, y los criterios a seguir por los programadores a la hora de codificar, documentar y probar, también es objetivo de esta fase.

El análisis orgánico afecta a:

1. *Los datos*

Es preciso hacer una correspondencia entre las especificaciones pedidas por los usuarios y los datos integrantes del sistema. Como resultado de este estudio se determinarán la asignación de ficheros y las relaciones entre unos y otros. Es muy diferente utilizar una base de datos de tipo relacional que un sistema de ficheros indexados, por lo que esta etapa es importante y decisiva en el resto del proceso de análisis.

Por cada archivo que utilice la aplicación, se deberá generar la siguiente documentación:

— descripción de archivos,
— diseño gráfico de registros,
— descripción de registros.

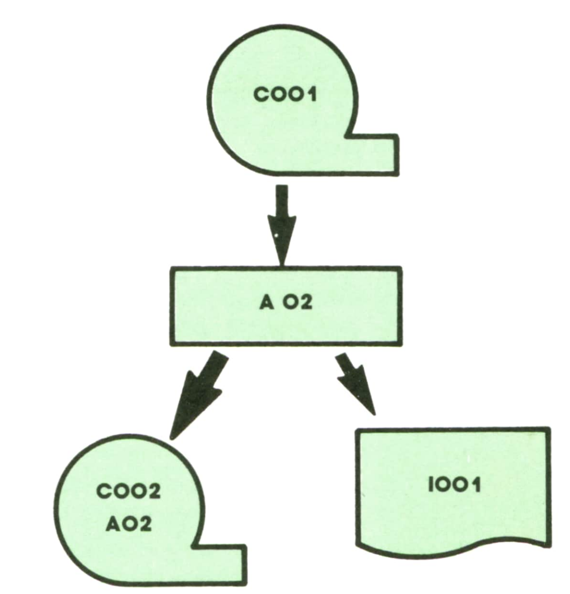
2. *Los programas*

Al igual que en el caso anterior, el trabajo a realizar puede considerarse como una continuación a la fase de análisis funcional.

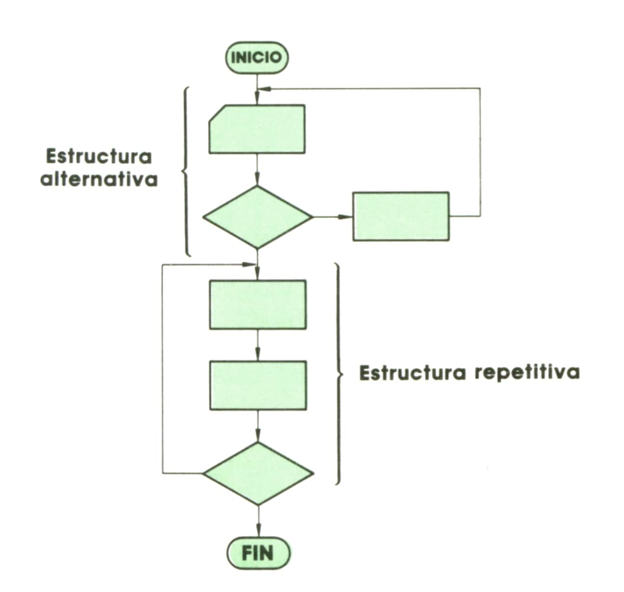
En este punto se analizará uno por uno cada proceso, determinando los programas necesarios para cada caso. Al final se explicarán los programas comunes a varios procesos.

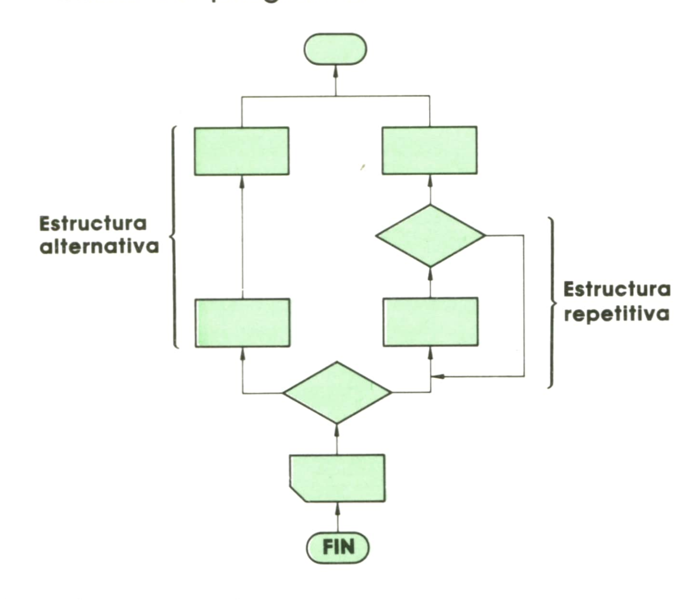
Por cada programa habrá que especificar:

— su descripción general y los objetivos,
— descripción detallada a nivel de rutinas,
— organigrama de funcionamiento y tablas de decisión,
— conexiones con otros programas,
— entradas al programa,
— diseños de informes o listados, producto del programa.



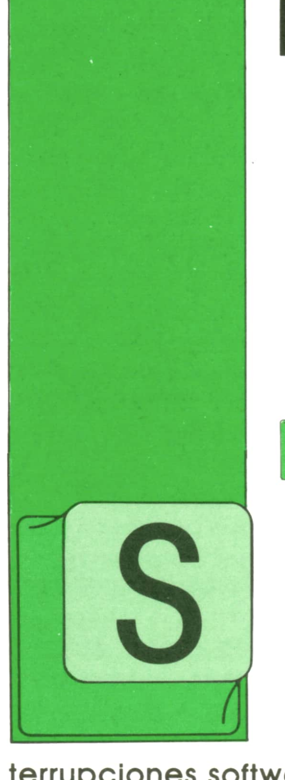

Organigrama de una unidad de tratamiento.

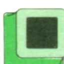





Estructuras básicas en programación: estructura alternativa es la que ejecuta un conjunto de operaciones sólo una vez de acuerdo a la condición analizada y estructura repetitiva en la que ejecutan las operaciones hasta que se cumpla una cierta condición.

MAQUINA 8088





Instrucciones INT e IRET

SON dos instrucciones ejecutables (análogas a las instrucciones CALL y RET anteriormente descritas) que sirven para comenzar y terminar la ejecución de las interrupciones software.

Para ejecutar una interrupción software se emplea la instrucción INT, abreviatura de «interrupt». Para indicar que el control de ejecución debe volver a la instrucción siguiente a la que llamó a la interrupción, se emplea la instrucción IRET, abreviatura de «interrupt return», que significa «retorno de la interrupción».

El mecanismo que permite que cuando se termina una interrupción se vuelva a la instrucción adecuada, es el siguiente:

Cuando se produce una interrupción de cualquier tipo, el 8088 carga en la pila de ejecución (STACK) tres palabras: las dos primeras son la dirección a la que debe volver (desplazamiento y registro de segmento) y la tercera es el estado de los flags. A continuación bifurca al código asociado a la interrupción definido en la tabla denominada «vector de interrupciones».

Inversamente, cuando se ejecuta una instrucción IRET, el 8088 descarga de la pila, primero el estado de los flags y después la dirección de retorno y bifurca a esa dirección.

Los formatos de estas instrucciones son los siguientes:

```
[etiqueta1:]    INT     numero

[etiqueta2:]    IRET
```

Etiqueta 1 y etiqueta 2. Son nombres que se pueden utilizar opcionalmente para identificar a estas instrucciones (como ocurre con todas las instrucciones ejecutables).

INT e IRET. Son los códigos de operación.

número. Es el operando por el que se define a la instrucción INT cuál es la interrupción que se quiere ejecutar. Puede ser cualquier número que pueda contenerse en un byte (es decir, comprendido entre 0 y 255) y puede expresarse, como las demás constantes numéricas, en decimal, hexadecimal, binario, etc., aunque lo más frecuente es que se haga en hexadecimal (terminando el código con una «H»).

Los códigos que se ejecutan con las interrupciones software desempeñan conceptualmente el mismo papel que las rutinas y se escriben utilizando las mismas técnicas (ver tomo 19), por lo que se les puede llamar «rutinas asociadas a interrupción» o simplemente «rutinas», pero no hay que olvidar las diferencias entre éstas y las rutinas verdaderas:

— Las rutinas se llaman con instrucciones CALL y las interrupciones con INT.

— Las rutinas se terminan con RET y las interrupciones con IRET.

— El código que atiende a cada interrupción debe tener un apuntador en la posición adecuada de la tabla denominada «vector de interrupciones» (ver tomo 25), mientras que las rutinas no tienen ningún requerimiento de este tipo.

— Como contrapartida, las rutinas tienen que ser llamadas desde el programa ejecutable que las contiene (aunque

pueden estar en módulos fuente diferentes que luego se ensamblan y se enlazan con el programa LINK), mientras que las interrupciones pueden ser llamadas desde programas diferentes.

En el caso en que se llame a una interrupción desde un programa diferente al que contiene el código que atiende a la interrupción (que es el caso más frecuente), dicho código debe residir permanentemente en memoria.

Se dice que un programa «reside en memoria» cuando está en memoria ROM o cuando está en una zona de memoria RAM respetada por el Sistema Operativo. Es decir, una zona de memoria que no utilizará para cargar otros programas.

La memoria ROM conserva la información aunque se interrumpa la alimentación eléctrica, mientras que la memoria RAM pierde su contenido cuando desaparece por un momento la alimentación.



Instrucciones CLI y STI

Son dos instrucciones que modifican el estado del flag de control de interrupciones IF («interrup enable flag»). Estas instrucciones no tienen operandos y, por tanto, se utilizan escribiendo únicamente su código de operación.

CLI pone en «off» (0) el flag IF sin afectar a los demás flags. En esta situación el 8088 no reconoce las interrupciones externas que le llegan por el hilo INTR desde los periféricos. Sin embargo, este flag no afecta a las interrupciones internas (INT, INTO, etc.) ni a las interrupciones externas que llegan al 8088 por el hilo NMI.

STI pone en «on» (1) el flag IF sin afectar a los demás flags. En esta situación el 8088 reconoce todas las interrupciones, tanto las internas como las externas.



Interrupciones software del sistema

El IBM/PC aprovecha el mecanismo de interrupciones software que tiene el 8088 para tener siempre en memoria una serie de rutinas que realizan funciones de utilidad general y que pueden ser llamadas desde cualquier programa, intercambiando información a través de los registros del microprocesador.

En la mayoría de los casos, las interrupciones son capaces de ejecutar varias funciones diferentes. Se ha estandarizado que el código de la función deseada se defina en el registro AH antes de llamar a la interrupción.

Las interrupciones software disponibles en el IBM/PC pertenecen a dos grupos:

— Las interrupciones del BIOS.

BIOS son las siglas de «Basic Input Output System» y se puede traducir por «Sistema básico de entradas y salidas». Las interrupciones de este grupo residen permanentemente en la zona de memoria ROM.

— Las interrupciones del DOS.

DOS son las siglas de «Disk Operating System», que significa «Sistema Operativo en Disco». Este grupo de interrupciones se carga en memoria RAM durante la fase de IPL (carga del programa inicial) y permanecen mientras se funcione con el DOS.

Las interrupciones de este grupo utilizan normalmente las interrupciones del grupo anterior. Por ejemplo, hay una función del DOS que permite representar en pantalla cadenas de caracteres. Pues bien, para representar cada uno de los caracteres de la cadena, dicha función DOS hace repetidas llamadas a una función del BIOS que representa los caracteres de uno en uno.



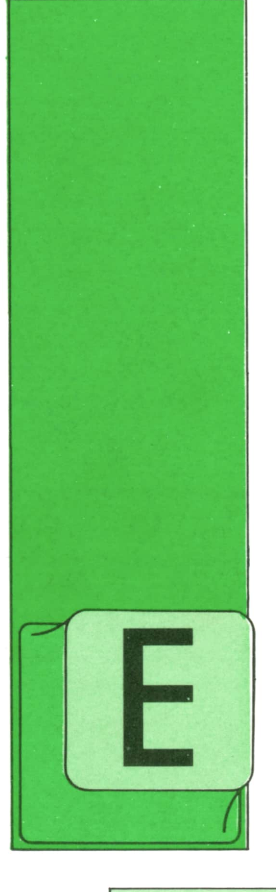
Interrupciones del BIOS

Como se puede ver en la tabla del tomo 25, el BIOS proporciona una serie de funciones asociadas a las interrupciones 16 a 26 (10H a 1AH); con ellas se pueden usar eficazmente los periféricos (teclado, pantallas, impresoras, discos, diskettes, etc.) sin tener que conocer las particularidades del manejo de cada uno de ellos. También se puede conocer la configuración de la máquina, es decir, la memoria y los periféricos instalados y otras funciones auxiliares como la definición y obtención de la fecha y hora del sistema, etc.

Las funciones del BIOS se usan fundamentalmente para programar los sistemas operativos como el DOS, pero pueden utilizarse desde cualquier programa escrito en ensamblador o en cualquier lenguaje que permita definir registros y ejecutar instrucciones INT.

PROGRAMAS

EDUCATIVOS • DE UTILIDAD • DE GESTION • DE JUEGOS



Programa: Editor de pantalla para IBM

EN tomos anteriores vimos un programa que nos permitía escribir textos con nuestro ordenador e imprimirlos más tarde con la ayuda de una impresora. En este tomo vamos a ver un programa que nos permite hacer lo mismo pero en un ordenador IBM pc, que trabaje bajo GWBASIC, o cualquier compatible.

```
1000 REM *********************************
1001 REM *                               *
1002 REM *          EDITOR               *
1003 REM *                               *
1004 REM *  POR: PETER BERGMANN          *
1005 REM *                               *
1006 REM *********************************
1007 REM
1008 REM **************************
1009 REM * (c) Ed. Siglo Cultural *
1010 REM * (c) 1987.              *
1011 REM **************************
1012 REM
1013 REM *** PRESENTACION ***
1014 REM
1015 KEY OFF
1016 CLS
1017 PRINT STRING$(79,"*")
1018 FOR I = 1 TO 19
1019    PRINT  "*"; TAB(79); "*"
1020 NEXT I
1021 PRINT STRING$(79,"*")
1022 LOCATE 8,36: PRINT "EDITOR"
1023 LOCATE 9,35: PRINT "--------"
1024 LOCATE 12,26: PRINT "(c) Ed. Siglo Cultural, 1987"
1025 LOCATE 24,1
1026 FOR I = 1 TO 1200: NEXT I
1027 REM
1028 REM *** DEFINE MEMORIA ***
1029 REM
1030 DEF SEG=&H9C40
1031 REM
1032 REM *** DECLARACION DE ARRAYS ***
1033 REM
```

```
1034 OPTION BASE 1
1035 DIM ROWMEM(200)
1036 DIM ROWEND(200)
1037 DIM BUFF$(1440)
1038 DIM FILE.NAME$(25)
1039 DIM FILE.LEN(25)
1040 DIM PBUFF$(80)
1041 REM*
1042 REM    *** DECLARATION DE FUNCIONES ***
1043 REM
1044 REM
1045 DEF FNROW(X,P) = X+(P-1)*20
1046 REM
1047 REM
1048 REM ***************************************
1049 REM *                 MENU                *
1050 REM ***************************************
1051 REM
1052 CLS
1053 LOCATE 3,32: PRINT "MENU DE ARCHIVO"
1054 LOCATE 4,30: PRINT "-----------------"
1055 LOCATE 6,29: PRINT "A - ABRIR ARCHIVO"
1056 LOCATE 7,29: PRINT "P - IMPRMIR ARCHIVO"
1057 LOCATE 8,29: PRINT "R - RENOMBRAR ARCHIVO"
1058 LOCATE 9,29: PRINT "C - COPIAR ARCHIVO"
1059 LOCATE 10,29: PRINT "K - BORRAR ARCHIVO"
1060 LOCATE 11,29: PRINT "X - SALIR AL SISTEMA"
1061 LOCATE 14,1: PRINT "(QUE OPCION DESEAS";
1062 INPUT O$
1063 IF (O$ <> "A") AND (O$ <> "P") AND (O$ <> "R") AND (O$ <> "K") AND (O$ <>
C") AND (O$ <> "X") THEN GOTO 1061
1064 IF O$ = "A" THEN GOSUB 1082
1065 IF O$ = "P" THEN GOSUB 1095
1066 IF O$ = "R" THEN GOSUB 1816
1067 IF O$ = "C" THEN GOSUB 1841
1068 IF O$ = "K" THEN GOSUB 1867
1069 IF O$ = "X" THEN GOTO 1071
1070 GOTO 1047
1071 REM
1072 REM *** SALIDA ***
1073 REM
1074 CLS
1075 PRINT "ADIOS..."
1076 FOR I=1 TO 10
1077    PRINT
1078 NEXT I
1079 KEY ON
1080 END
1081 GOTO 1080
1082 REM
1083 REM *************************************
1084 REM *      ABRIR Y EDITAR ARCHIVO       *
1085 REM *************************************
1086 REM
1087 CLS
1088 LET PAGE = 1
1089 LET TLEN = 5
1090 GOSUB 1110
1091 GOSUB 1119
1092 GOSUB 1168
1093 GOSUB 1227
1094 RETURN
1095 REM
1096 REM ************************************
1097 REM *      IMPRIMIR       TEXTO        *
1098 REM ************************************
1099 REM
```

```
1100 CLS
1101 GOSUB 1119
1102 IF ESFILE$ = "S" THEN GOTO 1108
1103 LOCATE 10,1: PRINT "ESE FICHERO NO EXISTE - (QUIERES OTRA VEZ:";
1104 INPUT R$
1105 IF R$ = "S" THEN GOTO 1101
1106 IF R$ <> "N" THEN GOTO 1103
1107 GOTO 1109
1108 GOSUB 1968
1109 RETURN
1110 REM
1111 REM *** INICIALIZACION DE ARRAYS ***
1112 REM
1113 PRINT "ESPERE, POR FAVOR...  (ARRAY INICIALIZACION)"
1114 FOR I = 1 TO 200
1115    LET ROWMEM (I) = 0
1116    LET ROWEND (I) = 0
1117 NEXT I
1118 RETURN
1119 REM
1120 REM *** NOMBRE DEL FICHERO A USAR ***
1121 REM
1122 CLS
1123 LET MEMLOC = 1!
1124 LET ENDTEXT$ = "N"
1125 LET ESFILE$ = "S"
1126 GOSUB 1919
1127 LOCATE 4,1: PRINT "(CUAL ES EL NOMBRE DEL FICHERO? ";
1128 GOSUB 1946
1129 LET FILE$ = INFILE$
1130 LOCATE 6,1: PRINT "EL NOMBRE ES ";FILE$
1131 ON ERROR GOTO 1135
1132 BLOAD FILE$
1133 ON ERROR GOTO 0
1134 RETURN
1135 REM
1136 REM *** FILE ERROR ***
1137 REM
1138 LET ESFILE$ = "N"
1139 LET ENDTEXT$ = "S"
1140 RESUME 1133
1141 REM
1142 REM *** GRABACION ***
1143 REM
1144 LET PAGE = 1: LET ROW = 1
1145 CLS
1146 LOCATE 1,1: PRINT "ESPERE POR FAVOR... (GRABACION)"
1147 GOSUB 1156
1148 POKE TOTAL, 234
1149 TOTAL = TOTAL + 1
1150 PRINT :PRINT PAGE,ROW,TOTAL
1151 BSAVE FILE$,0,TOTAL
1152 GOSUB 1801
1153 LOCATE 10,10: PRINT "TEXTO GRABADO EN ";FILE$
1154 LOCATE 12,10: PRINT "(ENTER PARA CONTINUAR)";: C$ = INPUT$(1)
1155 RETURN
1156 REM*
1157 REM*   ENCONTRAR TOTAL
1158 REM*
1159 LET TOTAL = 0
1160 LET I = 1
1161 LET INCHAR = 0
1162 WHILE INCHAR <> 234
1163    LET INCHAR = PEEK(I)
1164    TOTAL = TOTAL + 1
1165    I = I + 1
1166 WEND
```

```
1167 RETURN
1168 REM
1169 REM *** IMPRIMIR PAGINA ***
1170 REM
1171 SCREEN 0,0,1,1
1172 CLS
1173 LOCATE 12,23: PRINT "ESPERE PARA PAGINA ";PAGE;", POR FAVOR..."
1174 SCREEN 0,0,0,1
1175 CLS
1176 IF ENDTEXT$ = "S" THEN GOTO 1200
1177 ROWNUM = FNROW(1,PAGE)
1178 FOR ROW = 1 TO 20
1179     IF ENDTEXT$ = "S" THEN GOTO 1196
1180     LET ENDLINE$ = "N"
1181     LET ROWMEM(ROWNUM) = MEMLOC
1182     FOR COL = 1 TO 80
1183         IF ENDLINE$ = "S" THEN GOTO 1191
1184         IF ENDTEXT$ = "S" THEN GOTO 1191
1185         LET INCHAR = PEEK(MEMLOC)
1186         IF INCHAR = 234 THEN LET ENDTEXT$ = "S": MEMLOC = MEMLOC-1: GOTO 1191
1187         IF INCHAR = 20 THEN LET ENDLINE$ = "S": GOSUB 1219
1188         LOCATE ROW,COL: PRINT CHR$(INCHAR);
1189         IF INCHAR = 20 THEN PRINT CHR$(13);
1190         MEMLOC = MEMLOC + 1
1191     NEXT COL
1192     IF ENDTEXT$ = "N" THEN GOSUB 1211: REM CHECK WRAPAROUND
1193     LET ROWEND(ROWNUM) = MEMLOC - ROWMEM(ROWNUM) + 1
1194     MEMLOC = MEMLOC + 1
1195     ROWNUM = ROWNUM + 1
1196 NEXT ROW
1197 IF ENDTEXT$ = "N" THEN LET ROWMEM(ROWNUM) = MEMLOC
     ELSE LET ROWMEM(ROWNUM) = 0
1198 IF DLET$ = "S" THEN GOSUB 1612: GOTO 1207
1199 IF PRNT$ = "S" THEN GOSUB 1729: GOTO 1207
1200 LOCATE 21,1: PRINT STRING$(80,"_")
1201 LOCATE 22,1: PRINT CHR$(24);CHR$(25);CHR$(26);CHR$(27);" cursor mvmt"
1202 LOCATE 22,25: PRINT "Ctrl T - TAB SET"
1203 LOCATE 23,25: PRINT "Ctrl P - PRT SPC"
1204 LOCATE 22,50: PRINT "Ctrl Q - QUIT    "
1205 LOCATE 23,50: PRINT "Ctrl S - SAVE    "
1206 LOCATE 24,50: PRINT "Ctrl X - EXIT&SAVE";
1207 SCREEN 0,0,0,0
1208 LOCATE 1,1,1
1209 LET ROW = 1
1210 RETURN
1211 REM
1212 REM *** CHECK WRAPAROUND ***
1213 REM
1214 MEMLOC = MEMLOC - 1
1215 IF INCHAR = 20 THEN GOTO 1218
1216 IF INCHAR = 32 THEN GOTO 1218
1217 GOSUB 1347
1218 RETURN
1219 REM
1220 REM *** CHECK FOR EOF ***
1221 REM
1222 LET MEMLOC = MEMLOC + 1
1223 LET N = PEEK(MEMLOC)
1224 IF N = 234 THEN LET ENDTEXT$ = "S": MEMLOC = MEMLOC - 1
1225 MEMLOC = MEMLOC - 1
1226 RETURN
1227 REM
1228 REM *** PANTALLA DE ENTRADA ***
1229 REM
1230 LET QUIT$ = "N"
1231 ROW = CSRLIN
1232 COL = POS(0)
```

```
1233 LOCATE 25,67: PRINT "R=";ROW;" C=";COL
1234 LOCATE ROW,COL
1235 ROWNUM = FNROW(ROW,PAGE)
1236 A$ = INKEY$: IF A$ = "" THEN GOTO 1236
1237 LET ROW = CSRLIN
1238 LET COL = POS(0)
1239 IF (A$ > CHR$(31)) AND (A$ < CHR$(123)) THEN GOSUB 1268: GOTO 1231
1240 IF LEN(A$) = 2 THEN GOSUB 1257: GOTO 1231
1241 GOSUB 1244
1242 IF QUIT$ = "N" THEN GOTO 1231
1243 RETURN
1244 REM
1245 REM *** DECODE KEY IN ***
1246 REM
1247 IF (A$ = CHR$(168)) OR (A$ = CHR$(173)) OR (A$ = CHR$(128)) THEN GOSUB 1268
1248 IF A$ = CHR$(9) THEN GOSUB 1392: REM TAB
1249 IF A$ = CHR$(16) THEN GOSUB 1723: REM PRINT SPECIALTIES
1250 IF A$ = CHR$(20) THEN GOSUB 1407: REM TAB SET
1251 IF A$ = CHR$(17) THEN GOSUB 1358: REM QUIT
1252 IF A$ = CHR$(19) THEN GOSUB 1372: REM SAVE
1253 IF A$ = CHR$(24) THEN GOSUB 1380: REM SAVE EXIT
1254 IF A$ = CHR$(8) THEN GOSUB 1385: REM BACKSPACE
1255 IF A$ = CHR$(13) THEN GOSUB 1456: REM RETORNO DE CARR
1256 RETURN
1257 REM
1258 REM *** MOVIMIENTO DEL CURSOR ***
1259 REM
1260 IF MID$(A$,2,1)="H" THEN GOSUB 1423
1261 IF MID$(A$,2,1)="P" THEN GOSUB 1434
1262 IF MID$(A$,2,1)="K" THEN GOSUB 1450
1263 IF MID$(A$,2,1)="M" THEN GOSUB 1442
1264 IF DLET$ = "S" THEN GOTO 1267
1265 IF MID$(A$,2,1)="R" THEN GOSUB 1463
1266 IF MID$(A$,2,1)="S" THEN GOSUB 1606
1267 RETURN
1268 REM
1269 REM ***  GRABAR EN MEMORIA ***
1270 REM
1271 IF (COL > ROWEND(ROWNUM)) AND (ROWMEM(ROWNUM+1) > 0) THEN GOTO 1285
1272 IF (ROWNUM=1) AND (COL=1) THEN LET ROWMEM(ROWNUM) = 1: GOTO 1275
1273 IF COL = 1 THEN LET ROWMEM(ROWNUM) = ROWMEM(ROWNUM-1) + ROWEND(ROWNUM-1)
1274 IF (ROW = 20) AND (COL = 80) AND (PAGE = 10) THEN GOTO 1285
1275 MEMLOC = ROWMEM(ROWNUM) + COL - 1
1276 LOCATE 25,5: PRINT MEMLOC: LOCATE ROW,COL
1277 POKE MEMLOC, ASC(A$)
1278 IF (ROWMEM(ROWNUM+1) = 0) AND (COL > ROWEND(ROWNUM))
          THEN POKE MEMLOC + 1, 234
1279 IF COL > ROWEND(ROWNUM) THEN LET ROWEND(ROWNUM) = COL
1280 IF (COL = 80) AND (ROW = 20) THEN GOSUB 1286: LET ROW = 20
1281 IF (COL = 80) AND (A$ <> CHR$(32)) THEN GOSUB 1293
1282 PRINT A$;
1283 IF (A$ = CHR$(20)) AND (ROW = 20) THEN GOSUB 1286: GOTO 1285
1284 IF A$ = CHR$(20) THEN PRINT CHR$(13);
1285 RETURN
1286 REM*
1287 REM*  CLEAN PAGE
1288 REM*
1289 LET ENDTEXT$ = "S"
1290 IF PAGE < 10 THEN LET PAGE = PAGE + 1: GOSUB 1168: GOTO 1292
1291 LOCATE 25,1: PRINT "NO PAGINAS MAS";CHR$(7);
1292 RETURN
1293 REM
1294 REM *** EDICION DE LINEA (PARA GRABAR) ***
1295 REM
1296 GOSUB 1335
1297 K = 80 - I + 1
1298 IF ROW = 20 THEN GOTO 1303
```

```
1299 LOCATE ROW,K
1300 FOR J = K TO 80
1301     PRINT " ";
1302 NEXT J
1303 IF ROW = 20 THEN LOCATE 1,1 ELSE LOCATE ROW+1,1
1304 FOR J = 1 TO I - 2
1305     LET INCHAR = PEEK(MEMLOC + J + 1)
1306     PRINT CHR$(INCHAR);
1307 NEXT
1308 COL = POS(0)
1309 ROWEND(ROWNUM) = K
1310 ROWMEM(ROWNUM+1) = ROWMEM(ROWNUM) + ROWEND(ROWNUM)
1311 ROWEND(ROWNUM+1) = COL
1312 RETURN
1313 REM
1314 REM *** UNA PAGINA HACIA ATRAS ***
1315 REM
1316 IF PAGE = 1 THEN GOTO 1322
1317 PAGE = PAGE - 1
1318 ROWNUM = FNROW(1,PAGE)
1319 LET MEMLOC = ROWMEM(ROWNUM)
1320 LET ENDTEXT$ = "N"
1321 GOSUB 1168
1322 RETURN
1323 REM
1324 REM *** UNA PAGINA HACIA ADELANTE ***
1325 REM
1326 IF PAGE = 10 THEN GOTO 1334
1327 ROWNUM = FNROW(1,PAGE)
1328 IF ROWMEM(ROWNUM+1) = 0 THEN GOTO 1334
1329 PAGE = PAGE + 1
1330 ROWNUM = FNROW(1,PAGE)
1331 LET MEMLOC = ROWMEM(ROWNUM)
1332 LET ENDTEXT$ = "N"
1333 GOSUB 1168
1334 RETURN
1335 REM
1336 REM *** BUSCAR PARA ESPACIO ***
1337 REM
1338 LET SPCFND$ = "N"
1339 LET I = 0
1340 WHILE SPCFND$ = "N"
1341     LET INCHAR = PEEK(MEMLOC)
1342     IF INCHAR = 32 THEN LET SPCFND$ = "S"
1343     MEMLOC = MEMLOC - 1
1344     I = I + 1
1345 WEND
1346 RETURN
1347 REM
1348 REM *** EDICION DE LINEA (PARA IMPRIMIR) ***
1349 REM
1350 GOSUB 1335
1351 K = 80 - I + 1
1352 LOCATE ROW,K
1353 FOR J = K TO 80
1354     PRINT " ";
1355 NEXT J
1356 MEMLOC = MEMLOC + 1
1357 RETURN
1358 REM
1359 REM *** QUIT ***
1360 REM
1361 SCREEN 0,0,1,1
1362 CLS
1363 LOCATE 2,34: PRINT "*** QUIT ***";CHR$(7)
1364 LOCATE 5,26: PRINT "-AVISO - TEXTO WILL BE LOST!"
1365 LOCATE 7,26: PRINT "(ESTAS SEGURO (S/N)";
1366 INPUT R$
```

```
1367 IF R$ = "S" THEN LET QUIT$ = "S": GOTO 1371
1368 IF R$ <> "N" THEN GOTO 1365
1369 SCREEN 0,0,0,0
1370 LOCATE 1,1,1
1371 RETURN
1372 REM
1373 REM *** GRABAR TEXTO ***
1374 REM
1375 SCREEN 0,0,1,1
1376 GOSUB 1141: REM * SAVE TEXT
1377 SCREEN 0,0,0,0
1378 LOCATE 1,1,1
1379 RETURN
1380 REM
1381 REM *** GRABAR TEXTO Y SALIDA ***
1382 GOSUB 1141
1383 LET QUIT$ = "S"
1384 RETURN
1385 REM
1386 REM *** BACKSPACE ***
1387 REM
1388 PRINT CHR$(29);: PRINT " ";: PRINT CHR$(29);
1389 MEMLOC = ROWMEM(ROWNUM) + COL - 2
1390 POKE MEMLOC, 32
1391 RETURN
1392 REM
1393 REM *** TAB ***
1394 REM
1395 IF (ROWMEM(ROWNUM) <> 0) OR (COL <> 1) THEN GOTO 1404
1396 IF ROW = 1 THEN LET MEMLOC = 1 ELSE MEMLOC = ROWMEM(ROWNUM-1) + ROWEND(ROWN
UM-1)
1397 LET ROWMEM(ROWNUM) = MEMLOC: LET ROWEND(ROWNUM) = TLEN
1398 FOR I = 1 TO TLEN
1399    POKE MEMLOC,32
1400    MEMLOC = MEMLOC + 1
1401 NEXT I
1402 LOCATE ROW,TLEN
1403 GOTO 1406
1404 MEMLOC = ROWMEM(ROWNUM) + COL
1405 IF ROWEND(ROWNUM) > COL + TLEN THEN LOCATE ROW,COL+TLEN ELSE LOCATE ROW,ROW
END(ROWNUM)
1406 RETURN
1407 REM
1408 REM *** TAB SET ***
1409 REM
1410 LOCATE 25,1: PRINT "INPUT TAB LENGTH: ";
1411 ON ERROR GOTO 1418
1412 INPUT TLEN
1413 ON ERROR GOTO 0
1414 IF TLEN > 50 THEN GOTO 1410
1415 LOCATE 25,1: PRINT SPACE$(35)
1416 LOCATE ROW,COL
1417 RETURN
1418 REM
1419 REM *** TAB SET ERROR ***
1420 REM
1421 LOCATE 25,1: PRINT SPACE$(35)
1422 RESUME 1410
1423 REM
1424 REM *** CURSOR ARRIBA ***
1425 REM
1426 IF (ROW = 1) AND (PAGE = 1) THEN GOTO 1433
1427 IF ROW > 1 THEN GOTO 1430
1428 GOSUB 1313: REM PAGE BACK
1429 LET ROW = 21
1430 LET ROW = ROW-1
1431 ROWNUM = FNROW(ROW,PAGE)
```

```
1432 IF ROWEND(ROWNUM) > COL THEN LOCATE ROW,COL ELSE LOCATE ROW,ROWEND(ROWNUM)
1433 RETURN
1434 REM
1435 REM *** CURSOR ABAJO ***
1436 REM
1437 IF (ROW = 20) AND (ROWMEM(ROWNUM+1) = 0) THEN GOTO 1441
1438 IF (ROW = 20) AND (ROWMEM(ROWNUM+1) <> 0) THEN GOSUB 1324: GOTO 1441
1439 IF ROWMEM(ROWNUM+1) = 0 THEN GOTO 1441
1440 IF ROWEND(ROWNUM+1) > COL THEN LOCATE ROW+1,COL ELSE LOCATE ROW+1,ROWEND(RO
WNUM+1)
1441 RETURN
1442 REM
1443 REM *** CURSOR DERECHO ***
1444 REM
1445 IF ROWEND(ROWNUM) > COL THEN PRINT CHR$(28);: GOTO 1449
1446 IF ROW = 20 THEN GOTO 1449
1447 IF ROWMEM(ROWNUM+1) <> 0 THEN LET ROW=ROW+1: LOCATE ROW,1: GOTO 1449
1448 IF ROWEND(ROWNUM) = COL THEN PRINT CHR$(28);
1449 RETURN
1450 REM*
1451 REM*  CURSOR IZQUIERDO
1452 REM*
1453 IF COL > 1 THEN PRINT CHR$(29);: GOTO 1455
1454 IF ROW <> 1 THEN LOCATE ROW-1,ROWEND(ROWNUM-1)
1455 RETURN
1456 REM
1457 REM
1458 REM *** RETORNO DE CARRO ***
1459 REM
1460 LET A$ = CHR$(20)
1461 IF COL+1 > ROWEND(ROWNUM) THEN GOSUB 1268 ELSE GOSUB 1434
1462 RETURN
1463 REM*
1464 REM*  INSERT FUNCTION
1465 REM*
1466 LET INSERT$ = "S"
1467 START = ROWMEM(ROWNUM) + COL
1468 LET NUMIN = 0
1469 GOSUB 1479
1470 LET ROW = CSRLIN
1471 LET COL = POS(0)
1472 LOCATE 25,67: PRINT "R=";ROW;" C=";COL
1473 LOCATE ROW,COL
1474 A$ = INKEY$: IF A$ = "" THEN GOTO 1474
1475 IF (A$ > CHR$(31)) AND (A$ < CHR$(123)) THEN GOSUB 1506: GOTO 1470
1476 GOSUB 1538
1477 IF INSERT$ = "S" THEN GOTO 1470
1478 RETURN
1479 REM*
1480 REM*  BUILD INSERT SCREEN
1481 REM*
1482 LOCATE 25,1: PRINT "ESPERE PARA INSERT..."
1483 SCREEN 0,0,1,0
1484 CLS
1485 MEMLOC = ROWMEM(ROWNUM)
1486 FOR I = 1 TO COL - 1
1487      LET INCHAR = PEEK(MEMLOC)
1488      PRINT CHR$(INCHAR);
1489      MEMLOC = MEMLOC + 1
1490 NEXT I
1491 LET I = 1
1492 WHILE (I < 80) AND (INCHAR <> 234) AND (INCHAR <> 13)
1493      LET INCHAR = PEEK(MEMLOC)
1494      LOCATE 20,I: PRINT CHR$(INCHAR)
1495      MEMLOC = MEMLOC + 1
1496      I = 1 + 1
1497 WEND
```

```
1498 LOCATE 21,1: PRINT STRING$(80,"_")
1499 LOCATE 23,1: PRINT "I N S E R T  S C R E E N ";
1500 LOCATE 22,30: PRINT CHR$(24);CHR$(25);CHR$(27);CHR$(26);" - cursor mvmt";
1501 LOCATE 23,30: PRINT "Ctrl T - TAB SET";
1502 LOCATE 24,30: PRINT "Ctrl R - RECOMPOSE - EXIT";
1503 SCREEN 0,0,1,1
1504 LOCATE 1,COL,1
1505 RETURN
1506 REM*
1507 REM*    GRABACION EN INSERT ARRAY
1508 REM*
1509 NUMIN = NUMIN + 1
1510 IF NUMIN > 1440 THEN LOCATE 25,1: PRINT "BUFFER FULL";CHR$(7);: GOTO 1515
1511 LET BUFF$(NUMIN) = A$
1512 IF (COL = 80) AND (A$ <> " ") THEN  GOSUB 1516: GOTO 1515
1513 PRINT A$;
1514 IF A$ = CHR$(20) THEN PRINT CHR$(13);
1515 RETURN
1516 REM*
1517 REM*   EDICION DE LINEA PARA ARRAY
1518 REM*
1519 LET SPCFND$ = "N"
1520 LET I = 0
1521 WHILE SPCFND$ = "N"
1522      NUMIN = NUMIN - 1
1523      IF BUFF$(NUMIN) = " "  THEN LET SPCFND$ = "S"
1524      I = I + 1
1525 WEND
1526 LOCATE ROW+1,1
1527 FOR J = 1 TO I
1528    PRINT BUFF$(NUMIN + J);
1529 NEXT J
1530 K = 80 - I
1531 LOCATE ROW,K
1532 FOR J = K TO 80
1533    PRINT " ";
1534 NEXT J
1535 NUMIN = NUMIN + I
1536 LOCATE ROW+1,I+1
1537 RETURN
1538 REM*
1539 REM*   SPECIALITATES
1540 REM*
1541 IF (A$ = CHR$(168)) OR (A$ = CHR$(173)) OR (A$ = CHR$(128)) THEN GOSUB 1506
1542 IF A$ = CHR$(20) THEN GOSUB 1407
1543 IF A$ = CHR$(9) THEN GOSUB 1548
1544 IF A$ = CHR$(8) THEN GOSUB 1558
1545 IF A$ = CHR$(18) THEN GOSUB 1572
1546 IF A$ = CHR$(13) THEN GOSUB 1566
1547 RETURN
1548 REM*
1549 REM*   TAB
1550 REM*
1551 IF COL <> 1 THEN GOTO 1557
1552 FOR I = 1 TO TLEN
1553      NUMIN = NUMIN + 1
1554      LET BUFF$(NUMIN) = " "
1555      PRINT " ";
1556 NEXT I
1557 RETURN
1558 REM*
1559 REM*  BACKSPACE
1560 REM*
1561 IF NUMIN = 0 THEN GOTO 1565
1562 PRINT CHR$(29);: PRINT " ";: PRINT CHR$(29);
1563 LET BUFF$(NUMIN) = " "
1564 NUMIN = NUMIN - 1
1565 RETURN
```

```
1566 REM*
1567 REM*  RETORNO DE CARRO
1568 REM*
1569 LET A$ = CHR$(20)
1570 GOSUB 1506
1571 RETURN
1572 REM*
1573 REM*   RECOMPOSE (INSERT)
1574 REM*
1575 IF NUMIN = 0 THEN GOTO 1600
1576 LET LASTPAGE = PAGE
1577 LET ROW = 1: LET PAGE = 1
1578 LET NO$ = "S"
1579 CLS
1580 LOCATE 1,1: PRINT "ESPERE POR FAVOR... (RECOMPOSICION)"
1581 GOSUB 1156
1582 MEMLOC = TOTAL + 1
1583 IF MEMLOC < START + NUMIN THEN MEMLOC = START + NUMIN - 1: LET NO$ = "N"
1584 FOR I = START-1 TO TOTAL
1585      LET INCHAR = PEEK(I)
1586      POKE MEMLOC, INCHAR
1587      MEMLOC = MEMLOC + 1
1588 NEXT I
1589 LET MEMLOC = START
1590 FOR I = 1 TO NUMIN
1591      POKE START + I - 2, ASC(BUFF$(I))
1592 NEXT I
1593 IF NO$ = "N" THEN GOTO 1599
1594 MEMLOC = START + NUMIN - 2
1595 FOR I = 1 TO TOTAL - START + 2
1596      LET INCHAR = PEEK(TOTAL + I)
1597      POKE MEMLOC + I, INCHAR
1598 NEXT I
1599 LET PAGE = LASTPAGE
1600 LET INSERT$ = "N"
1601 LET ENDTEXT$ = "N"
1602 ROWNUM = FNROW(1,PAGE)
1603 LET MEMLOC = ROWMEM(ROWNUM)
1604 GOSUB 1168
1605 RETURN
1606 REM*
1607 REM*   DELETE FUNCTION
1608 REM*
1609 GOSUB 1612
1610 GOSUB 1626
1611 RETURN
1612 REM*
1613 REM*   BUILD DELETE SCREEN
1614 REM*
1615 FOR I = 22 TO 25
1616      LOCATE I,1: PRINT SPACE$(80);
1617 NEXT I
1618 LOCATE 23,1: PRINT "D E L E T E  S C R E E N";
1619 LOCATE 22,35: PRINT "Ctrl B - COMENZAR BLK";
1620 LOCATE 23,35: PRINT "Ctrl T - TERMINAR BLK";
1621 LOCATE 24,35: PRINT "Ctrl W - CAMBIAR BLK";
1622 LOCATE 22,60: PRINT "Ctrl R - RECOMPOSE";
1623 LOCATE 23,60: PRINT "Ctrl Q - ABORT - EXIT";
1624 LOCATE ROW,COL
1625 RETURN
1626 REM*
1627 REM*   SET FLAGS
1628 REM*
1629 LET STARTFLAG = 0: LET ENDFLAG = 0
1630 LET DLET$ = "S"
1631 LET INITPAGE = PAGE
1632 LET ROW = CSRLIN
1633 LET COL = POS(0)
```

```
1634 LOCATE 25,67: PRINT "R=";ROW;" C=";COL
1635 LOCATE ROW,COL
1636 ROWNUM = FNROW(ROW,PAGE)
1637 A$ = INKEY$: IF A$ = "" THEN GOTO 1637
1638 LOCATE 25,1: PRINT SPACE$(40): LOCATE ROW,COL
1639 IF LEN(A$) = 2 THEN GOSUB 1257 ELSE GOSUB 1642
1640 IF DLET$ = "S" THEN GOTO 1632
1641 RETURN
1642 REM*
1643 REM*  DECODE KEY IN
1644 REM*
1645 IF A$ = CHR$(2) THEN GOSUB 1651: REM BEGIN BLOCK
1646 IF A$ = CHR$(20) THEN GOSUB 1657: REM TERMINATE BLOCK
1647 IF A$ = CHR$(18) THEN GOSUB 1663: REM RECOMPOSE
1648 IF A$ = CHR$(17) THEN GOSUB 1694: REM QUIT
1649 IF A$ = CHR$(23) THEN GOSUB 1694: REM CAMBIAR
1650 RETURN
1651 REM*
1652 REM*   SET BEGIN FLAG
1653 REM*
1654 STARTFLAG = ROWMEM(ROWNUM) + COL - 1
1655 PRINT CHR$(26): LOCATE ROW,COL
1656 RETURN
1657 REM*
1658 REM*   SET END FLAG
1659 REM*
1660 ENDFLAG = ROWMEM(ROWNUM) + COL - 1
1661 PRINT CHR$(27): LOCATE ROW,COL
1662 RETURN
1663 REM*
1664 REM*   RECOMPOSE
1665 REM*
1666 LOCATE 25,1: PRINT SPACE$(30)
1667 IF STARTFLAG = 0 THEN LOCATE 25,1: PRINT "START FLAG NOT SET": GOTO 1692
1668 IF ENDFLAG = 0 THEN LOCATE 25,1: PRINT "END FLAG NOT SET": GOTO 1692
1669 IF ENDFLAG < STARTFLAG THEN LOCATE 25,1: PRINT "END MENOS QUE START": GOTO
1692
1670 LET ROW = 1: LET PAGE = 1
1671 CLS
1672 LOCATE 1,1: PRINT "ESPERE POR FAVOR... (RECOMPOSICION)"
1673 GOSUB 1156
1674 PRINT: PRINT STARTFLAG;ENDFLAG;TOTAL
1675 FOR I = ENDFLAG + 1 TO TOTAL
1676     LET INCHAR = PEEK(I)
1677     POKE STARTFLAG, INCHAR
1678 PRINT CHR$(INCHAR);
1679     STARTFLAG = STARTFLAG + 1
1680 NEXT I
1681 STOP
1682 LET DLET$ = "N"
1683 LET PAGE = INITPAGE
1684 LET ENDTEXT$ = "N"
1685 ROWNUM = FNROW(1,PAGE)
1686 LET MEMLOC = ROWMEM(ROWNUM)
1687 GOSUB 1168
1688 FOR I = ROWNUM TO 200
1689    LET ROWMEM(I) = 0: LET ROWEND(I) = 0:
1690 NEXT I
1691 LET COL = 1
1692 LOCATE ROW,COL
1693 RETURN
1694 REM*
1695 REM*  QUIT
1696 REM*
1697 IF A$ = CHR$(23) THEN GOTO 1703
1698 LET DLET$ = "N"
1699 FOR I = 22 TO 25
1700    LOCATE I,1: PRINT SPACE$(80);
```

```
1701 NEXT I
1702 GOSUB 1200
1703 IF STARTFLAG <> 0 THEN LET NUMIN = STARTFLAG: GOSUB 1707
1704 IF ENDFLAG <> 0 THEN LET NUMIN = ENDFLAG: GOSUB 1707
1705 LET STARTFLAG = 0: LET ENDFLAG = 0
1706 RETURN
1707 REM*
1708 REM*   RESET SCREEN
1709 REM*
1710 LET ROW = 1
1711 ROWNUM = FNROW(ROW,PAGE)
1712 LET FOUND$ = "N"
1713 WHILE FOUND$ = "N"
1714      IF NUMIN < ROWMEM(ROWNUM) THEN LET FOUND$ = "S"
1715      ROWNUM = ROWNUM + 1
1716 WEND
1717 ROWNUM = ROWNUM - 2
1718 COL = NUMIN - ROWMEM(ROWNUM) + 1
1719 INCHAR = PEEK(NUMIN)
1720 ROW = ROWNUM - (PAGE - 1) * 20
1721 LOCATE ROW,COL: PRINT CHR$(INCHAR)
1722 RETURN
1723 REM
1724 REM *** PRINT SPECIALTIES ***
1725 REM
1726 GOSUB 1729
1727 GOSUB 1747
1728 RETURN
1729 REM*
1730 REM*   BUILD PRINT SCREEN
1731 REM*
1732 FOR I = 22 TO 25
1733      LOCATE I,1: PRINT SPACE$(80);
1734 NEXT I
1735 LOCATE 23,1: PRINT "P R I N T";
1736 LOCATE 22,15: PRINT "Ctrl B - SET EMPHASIZE";
1737 LOCATE 23,15: PRINT "Ctrl T - CANCEL EMPHA";
1738 LOCATE 24,15: PRINT "Ctrl W - SET UNDERLN";
1739 LOCATE 25,15: PRINT "Ctrl R - CANCEL UNDRLN";
1740 LOCATE 22,42: PRINT "Ctrl S - SET ITALC";
1741 LOCATE 23,42: PRINT "Ctrl Z - CANCEL ITALIC";
1742 LOCATE 24,42: PRINT "Ctrl E - SET CENTER";
1743 LOCATE 25,42: PRINT "Ctrl D - CANCEL CENTER";
1744 LOCATE 25,1: PRINT "Ctrl X - EXIT";
1745 LOCATE ROW,COL
1746 RETURN
1747 REM*
1748 REM*   SET FLAGS
1749 REM*
1750 LET PRNT$ = "S"
1751 LET ROW = CSRLIN
1752 LET COL = POS(0)
1753 LOCATE 25,67: PRINT "R=";ROW;" C=";COL
1754 LOCATE ROW,COL
1755 ROWNUM = FNROW(ROW,PAGE)
1756 A$ = INKEY$: IF A$ = "" THEN GOTO 1756
1757 IF LEN(A$) = 2 THEN GOSUB 1257 ELSE GOSUB 1760
1758 IF PRNT$ = "S" THEN GOTO 1751
1759 RETURN
1760 REM*
1761 REM*  DECODE KEY IN
1762 REM*
1763 IF A$ = CHR$(24) THEN GOSUB 1788: GOTO 1778: REM EXIT
1764 GOSUB 1779
1765 IF ISSPACE$ = "N" THEN GOTO 1778
1766 LET CHAR = 0
1767 IF A$ = CHR$(2)  THEN LET CHAR = 1: REM BEGIN EMPH
```

```
1768 IF A$ = CHR$(20) THEN LET CHAR = 2: REM END EMPH
1769 IF A$ = CHR$(23) THEN LET CHAR = 220: REM BEGIN UNDRLN
1770 IF A$ = CHR$(18) THEN LET CHAR = 223: REM END UNDERLN
1771 IF A$ = CHR$(19) THEN LET CHAR = 221: REM SET ITALIC
1772 IF A$ = CHR$(26) THEN LET CHAR = 222: REM END ITALIC
1773 IF A$ = CHR$(5)  THEN LET CHAR = 200: REM SET CCENTER
1774 IF A$ = CHR$(4)  THEN LET CHAR = 201: REM END CENTER
1775 IF CHAR = 0 THEN GOTO 1778
1776 LOCATE ROW,COL: PRINT CHR$(CHAR);CHR$(29);
1777 POKE MEMNUM, CHAR
1778 RETURN
1779 REM
1780 REM *** CHECK POS ***
1781 REM
1782 LET ISSPACE$ = "N"
1783 ROWNUM = FNROW(ROW,PAGE)
1784 MEMNUM = ROWMEM(ROWNUM) + COL - 1
1785 LET INCHAR = PEEK(MEMNUM)
1786 IF INCHAR = 32 THEN LET ISSPACE$ = "S"
1787 RETURN
1788 REM
1789 REM *** EXIT ***
1790 REM
1791 LET PRNT$ = "N"
1792 FOR I = 22 TO 25
1793    PRINT SPACE$(80)
1794 NEXT I
1795 GOSUB 1200
1796 RETURN
1797 REM**********************************
1798 REM*          FILE MANAGEMENT          *
1799 REM**********************************
1800 REM*
1801 REM*
1802 REM*  ADD FILE NAME
1803 REM*
1804 LET EXIST$ = "N"
1805 FOR I = 1 TO NUMFILES
1806      IF FILE.NAME$(I) = FILE$ THEN LET EXIST$ = "S"
1807 NEXT I
1808 IF EXIST$ = "S" THEN GOTO 1815
1809 OPEN "FILES.ED" FOR APPEND AS #1
1810 WRITE #1, FILE$, TOTAL
1811 CLOSE #1
1812 NUMFILES = NUMFILES + 1
1813 LET FILE.NAME$(NUMFILES) = FILE$
1814 LET FILE.LEN(NUMFILES) = TOTAL
1815 RETURN
1816 REM*
1817 REM*  RENOMBRAR FICHERO
1818 REM*
1819 CLS
1820 GOSUB 1919
1821 LOCATE 1,32: PRINT "RENOMBRAR FICHERO"
1822 LOCATE 2,31: PRINT "___________________"
1823 LOCATE 4,1: PRINT "NOMBRE DE FICHERO CAMBIAR: ";
1824 GOSUB 1946
1825 LET FILE.O$ = INFILE$
1826 LOCATE 6,1: PRINT "NOMBRE NUEVO: ";
1827 GOSUB 1946
1828 LET FILE.N$ = INFILE$
1829 ON ERROR GOTO 1892
1830 BLOAD FILE.O$
1831 ON ERROR GOTO 0
1832 LET B$ = ".BAS"
1833 FILE.O.B$ = FILE.O$ + B$
1834 FILE.N.B$ = FILE.N$ + B$
```

```
1835 NAME FILE.O.B$ AS FILE.N.B$
1836 GOSUB 1899
1837 LOCATE 9,5: PRINT FILE.O$;" ES ";FILE.N$;" AHORA."
1838 LOCATE 10,5: PRINT "(ENTER A CONTINUAR)"
1839 LET C$ = INPUT$(1)
1840 RETURN
1841 REM*
1842 REM*  COPIAR FICHERO
1843 REM*
1844 CLS
1845 GOSUB 1919
1846 LOCATE 1,33: PRINT "COPIAR FICHERO"
1847 LOCATE 2,32: PRINT "________________"
1848 LOCATE 4,1: PRINT "NOMBRE DE FICHERO COPIAR: ";
1849 GOSUB 1946
1850 LET FILE.O$ = INFILE$
1851 LOCATE 6,1: PRINT "NOMBRE NUEVO: ";
1852 GOSUB 1946
1853 LET FILE.N$ = INFILE$
1854 ON ERROR GOTO 1892
1855 BLOAD FILE.O$
1856 ON ERROR GOTO 0
1857 LET I = 1
1858 WHILE FILE.NAME$(I) <> FILE.O$
1859     I = I + 1
1860 WEND
1861 BSAVE FILE.N$,0,FILE.LEN(I)
1862 GOSUB 1899
1863 LOCATE 9,5: PRINT FILE.O$;" ES ";FILE.N$;" AHORA."
1864 LOCATE 10,5: PRINT "(ENTER A CONTINUAR)"
1865 LET C$ = INPUT$(1)
1866 RETURN
1867 REM*
1868 REM*  BORRAR FICHERO
1869 REM*
1870 CLS
1871 GOSUB 1919
1872 LOCATE 1,33: PRINT "BORRAR FICHERO"
1873 LOCATE 2,32: PRINT "________________"
1874 LOCATE 4,1: PRINT "NOMBRE DE FICHERO BORRAR: ";
1875 GOSUB 1946
1876 LET FILE.O$ = INFILE$
1877 ON ERROR GOTO 1892
1878 BLOAD FILE.O$
1879 ON ERROR GOTO 0
1880 LET B$ = ".BAS"
1881 FILE.O.B$ = FILE.O$ + B$
1882 LOCATE 7,31: PRINT CHR$(7);CHR$(7);"(ESTAS SEGURO (S/N)";
1883 INPUT R$
1884 IF R$ = "N" THEN GOTO 1891
1885 IF R$ <> "S" THEN GOTO 1882
1886 KILL FILE.O.B$
1887 GOSUB 1899
1888 LOCATE 9,5: PRINT FILE.O$;" ES MUERTE AHORA."
1889 LOCATE 10,5: PRINT "(ENTER A CONTINUAR)"
1890 LET C$ = INPUT$(1)
1891 RETURN
1892 REM*
1893 REM* NO FICHERO
1894 REM*
1895 LOCATE 7,5: PRINT FILE.O$;" NO EXISTE."
1896 IF O$ = "R" THEN RESUME 1838
1897 IF O$ = "C" THEN RESUME 1864
1898 IF O$ = "K" THEN RESUME 1889
1899 REM*
1900 REM*  CAMBIAR FICHERO DE NOMBRES
1901 REM*
```

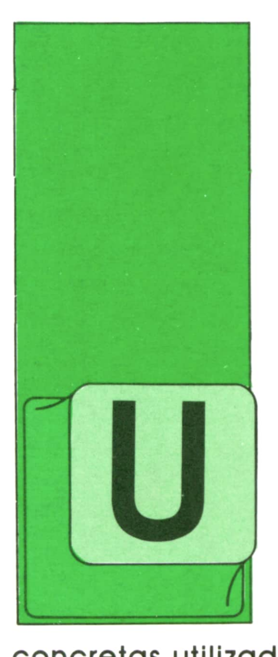
```
1902 FOR I = 1 TO NUMFILES
1903      IF FILE.NAME$(I) = FILE.O$ THEN LET INUM = I
1904 NEXT I
1905 IF O$ <> "C" THEN GOTO 1909
1906 NUMFILES = NUMFILES + 1
1907 LET FILE.NAME$(NUMFILES) = FILE.N$
1908 LET FILE.LEN(NUMFILES) = FILE.LEN(INUM)
1909 IF O$ = "R" THEN LET FILE.NAME$(INUM) = FILE.N$
1910 IF O$ = "K" THEN LET FILE.NAME$(INUM) = " "
1911 KILL "FILES.ED"
1912 OPEN "FILES.ED" FOR OUTPUT AS #1
1913 FOR J = 1 TO NUMFILES
1914    IF FILE.NAME$(J) = " " THEN GOTO 1916
1915    WRITE #1, FILE.NAME$(J), FILE.LEN(J)
1916 NEXT J
1917 CLOSE #1
1918 RETURN
1919 REM*
1920 REM*  LISTO DE FICHEROS
1921 REM*
1922 LOCATE 11,1: PRINT STRING$(80,"_")
1923 LOCATE 12,1: PRINT "FICHEROS: "
1924 LET COL = 1: ROW = 14
1925 LET I = 1
1926 ON ERROR GOTO 1940
1927 OPEN "FILES.ED" FOR INPUT AS #1
1928 ON ERROR GOTO 0
1929 WHILE NOT EOF(1)
1930    INPUT #1, FILE.NAME$(I), FILE.LEN(I)
1931    LOCATE ROW,COL: PRINT FILE.NAME$(I)
1932    COL = COL + 20
1933    IF COL > 61 THEN ROW = ROW + 1: LET COL = 1
1934    I = I + 1
1935 WEND
1936 CLOSE #1
1937 LET NUMFILES = I - 1
1938 IF NUMFILES = 0 THEN LOCATE 12,13: PRINT "NINGUNO"
1939 RETURN
1940 REM*
1941 REM*  NO HAY LISTO
1942 REM*
1943 LOCATE 12,13: PRINT "NINGUNO"
1944 LET NUMFILES = 0
1945 RESUME 1939
1946 REM*
1947 REM*    INPUT FILE NAME
1948 REM*
1949 LET INFILE$ = ""
1950 LET CHAR$ = INPUT$(1)
1951 IF (ASC(CHAR$) = 13) THEN GOTO 1958
1952 IF ASC(CHAR$) = 8 THEN GOSUB 1960: GOTO 1950
1953 IF (ASC(CHAR$) < 48) OR (ASC(CHAR$) > 90) THEN GOTO 1950
1954 IF (ASC(CHAR$) > 57) AND (ASC(CHAR$) < 65) THEN GOTO 1950
1955 INFILE$ = INFILE$ + CHAR$
1956 PRINT CHAR$;
1957 GOTO 1950
1958 IF INFILE$ = "" THEN GOTO 1950
1959 RETURN
1960 REM
1961 REM *** BACKSPACE ***
1962 REM
1963 INLEN = LEN(INFILE$)
1964 IF INLEN = 0 THEN GOTO 1967
1965 PRINT CHR$(29);" ";CHR$(29);
1966 INFILE$ = LEFT$(INFILE$,INLEN-1)
1967 RETURN
1968 REM
```

```
1969 REM***********************************
1970 REM*       IMPRIMIR FICHERO          *
1971 REM***********************************
1972 REM
1973 REM
1974 GOSUB 1978: REM BUILD SCREEN
1975 GOSUB 1991: REM INPUT PARAMETERS
1976 GOSUB 2011
1977 RETURN
1978 REM
1979 REM *** BUILD SCREEN ***
1980 REM
1981 CLS
1982 PRINT "INFORMACION DE IMPRIMIR PARA FICHERO: ";FILE$
1983 LOCATE 4,10: PRINT "TYPO                                 (CAMBIO?    A QUE"
1984 LOCATE 5,10: PRINT "______________________________       ________    _____"
1985 LOCATE 7,10: PRINT "LINEAS POR PAGINA (60)                  N";
1986 LOCATE 8,10: PRINT "DRAFT O NLQ (D/N)                       D";
1987 LOCATE 9,10: PRINT "PICA O ELITE (P/E)                      P";
1988 LOCATE 10,10:PRINT "CONDENSED                               N";
1989 LOCATE 15,10:PRINT "ESTA CORRECTO                           S";
1990 RETURN
1991 REM
1992 REM *** INPUT INFO ***
1993 REM
1994 LET LINES = 60: LET TYPO$ = "D": LET TYPE$ = "P": LET COND$ = "N"
1995 LOCATE 7,47: INPUT R$
1996 IF R$ = "S" THEN LOCATE 7,57: INPUT LINES
1997 LOCATE 8,47: INPUT R$
1998 IF R$ = "N" THEN LET TYPO$ = "N"
1999 LOCATE 9,47: INPUT R$
2000 IF R$ = "E" THEN LET TYPE$ = "E"
2001 LOCATE 10,47: INPUT R$
2002 IF R$ = "S" THEN LET COND$ = "S"
2003 GOSUB 1978
2004 LOCATE 7,57: PRINT LINES
2005 LOCATE 8,57: PRINT TYPO$
2006 LOCATE 9,57: PRINT TYPE$
2007 LOCATE 10,57: PRINT COND$
2008 LOCATE 15,47: INPUT R$
2009 IF R$ = "N" THEN GOSUB 1978: GOTO 1991
2010 RETURN
2011 REM
2012 REM *** PRINT FICHERO ***
2013 REM
2014 CLS
2015 PRINT "PULSO RETORNO QUANDO ESTAS LISTO..."
2016 INPUT R$
2017 LPRINT CHR$(27)+CHR$(64)
2018 LPRINT CHR$(27)+CHR$(67)
2019 IF TYPO$ = "D" THEN LPRINT CHR$(27)+CHR$(120)+CHR$(48) ELSE LPRINT CHR$(27)
+CHR$(120)+CHR$(49)
2020 IF TYPE$ = "E" THEN LPRINT CHR$(27)+CHR$(77) ELSE LPRINT CHR$(27)+CHR$(80)
2021 IF (TYPO$ = "D") AND (COND$ = "S") THEN LPRINT CHR$(15)
2022 LET PAGE = 1
2023 LET MEMLOC = 1
2024 LET ENDPRINT$ = "N"
2025 LET COUNT = 0
2026 PBUFF$ = ""
2027 WHILE ENDPRINT$ = "N"
2028    COUNT = COUNT + 1
2029    LET INCHAR = PEEK(MEMLOC)
2030    IF (INCHAR > 31) AND (INCHAR < 128) THEN LET PBUFF$=PBUFF$+CHR$(INCHAR)
              ELSE GOSUB 2036
2031    IF COUNT = 80 THEN GOSUB 2049
2032    MEMLOC = MEMLOC + 1
```

```
2033 WEND
2034 LPRINT PBUFF$
2035 RETURN
2036 REM
2037 REM  *** PRINT SPECIALTIES ***
2038 REM
2039 IF INCHAR = 20 THEN LPRINT PBUFF$:LET PBUFF$ = "":LET COUNT=0: GOTO 2048
2040 IF INCHAR = 1 THEN PBUFF$ = PBUFF$+CHR$(32)+CHR$(27)+CHR$(69)
2041 IF INCHAR = 2 THEN PBUFF$ = PBUFF$+CHR$(27)+CHR$(70)+CHR$(32)
2042 IF INCHAR = 220 THEN PBUFF$ = PBUFF$+CHR$(32)+CHR$(27)+CHR$(45)+"1"
2043 IF INCHAR = 223 THEN PBUFF$ = PBUFF$+CHR$(27)+CHR$(45)+"0"+CHR$(32)
2044 IF INCHAR = 221 THEN PBUFF$ = PBUFF$+CHR$(32)+CHR$(27)+CHR$(52)
2045 IF INCHAR = 222 THEN PBUFF$ = PBUFF$+CHR$(27)+CHR$(53)+CHR$(32)
2046 IF INCHAR = 234 THEN LET ENDPRINT$ = "S"
2047 IF INCHAR = 200 THEN GOSUB 2063
2048 RETURN
2049 REM
2050 REM *** WRAPAROUND ***
2051 REM
2052 WHILE LOOK$ <> " "
2053      LET LOOK$ = MID$(PBUFF$,COUNT,1)
2054      COUNT = COUNT -1
2055 WEND
2056 COUNT = COUNT + 1
2057 PBUFF$ = LEFT$(PBUFF$,COUNT)
2058 LPRINT PBUFF$
2059 MEMLOC = MEMLOC + COUNT - 80
2060 COUNT = 0
2061 PBUFF$ = ""
2062 RETURN
2063 REM
2064 REM *** CENTERING ***
2065 REM
2066 IF PBUFF$ > "" THEN LPRINT PBUFF$
2067 LET PBUFF$ = ""
2068 LET COUNT = 0
2069 WHILE INCHAR <> 201
2070      LET INCHAR = PEEK(MEMLOC)
2071      IF INCHAR = 201 THEN GOTO 2074
2072      COUNT = COUNT + 1
2073      IF (INCHAR > 31) AND (INCHAR < 127) THEN PBUFF$ = PBUFF$ +CHR$(INCHAR)
             ELSE GOSUB 2036
2074 WEND
2075 NUMSPACE = (80 - COUNT)/2
2076 PBUFF1$ = SPACE$(NUMSPACE) + PBUFF$
2077 LPRINT PBUFF1$
2078 LET COUNT = 0: LET PBUFF$ = "": LET PBUFF1$ = ""
2079 RETURN
```

TECNICAS DE ANALISIS

SISTEMAS DE CODIGOS



NA vez presentados los «métodos predefinidos de desarrollo» y vistos, anteriormente, los elementos básicos del diseño de sistemas, vamos a abordar algunas técnicas concretas utilizadas en el análisis de sistemas informáticos (procesos de codificación, diseño y manejo de ficheros, etcétera), así como las técnicas genéricas de control de proyectos (utilizables en el desarrollo de sistemas informáticos complejos).

Respecto de los sistemas de códigos en concreto, debemos decir que es uno de los elementos más utilizados en cualquier proceso informático por la propia naturaleza simplificadora, operativa y economizadora de medios de las técnicas informáticas. En efecto, en una aplicación informática «se codifica todo» (desde el propio nombre de la aplicación, de sus partes constituyentes y de los documentos en que se definen, hasta los menores elementos que intervienen en ella: ficheros, registros y sus campos, campos y áreas de un impreso de salida o de un formato de pantalla, etc.) y, aunque la capacidad del ordenador supera o minimiza numerosas deficiencias, conviene prestar alguna atención a la definición y aplicación de los códigos, para su máxima eficacia.



Características de los códigos

Los códigos que se utilicen deben adecuarse a una serie de características típicas para que podamos asegurar su eficacia. Estas características se refieren tanto al propio código como a su utilización. Respecto al código en sí, se suelen señalar como necesarias las siguientes características: conciso, claro, concreto, adecuado y flexible.

a) El código ha de ser conciso, porque su finalidad es representar de una manera cómoda a un elemento o entidad cualquiera y si es más complicado que la propia entidad a la que representa pierde su eficacia como tal código. Sin embargo, normalmente, la concisión se consigue sacrificando claridad, por lo que hay que buscar un compromiso entre estas dos características.

b) Por claridad del código entendemos la facilidad con que se puede reproducir y memorizar. En efecto, un código formado por ocho dígitos, por ejemplo, difícilmente será retenido en la memoria por los usuarios (excepto en las combinaciones más utilizadas, como sucede con los teléfonos). Por ello es útil dividir el código en zonas o intercalar signos de diferente tipo (números, letras, signos especiales, etc.) en la secuencia de caracteres que forman el código. Además, en la transcripción de los códigos complicados se cometen más errores que en la reproducción de los sencillos.

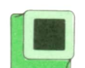
c) Es fundamental en el diseño y aplicación de un código, también, que éste sea concreto, no ambiguo. En efecto, si en la designación de entidades diversas para uso de las personas puede no ser tan crítica esta cualidad, porque las personas pueden suplir la ambigüedad con el resto de información que tienen del entorno o, en último caso, concretando mediante preguntas, en el caso de los códigos que deben ser utilizados por una máquina es fundamental la unicidad de cada código y la univocidad de su representación.

d) Es importante, asimismo, que se preste alguna atención al hecho de que el código se adecúe al uso que se ha de hacer de él. No tiene ningún sentido, por ejemplo, preparar un código detallado

de cada uno de los elementos de un conjunto si posteriormente sólo se va a procesar la información relativa a ellos por grupos o categorías. Por otro lado, no será igual un código que debe ser manejado por personas no habituadas al proceso de datos (personas, por tanto, que deberán utilizar códigos simples y llenos de significado para ellas) que si el código va a ser manejado casi exclusivamente dentro del proceso con ordenador o que si se busca específicamente que el código no sea fácilmente inteligible para las personas que incidentalmente lo puedan leer.

e) Por último, hemos de asegurarnos de que el código utilizado sea lo más flexible posible en cuanto a su evolución durante el período de vida de la aplicación o sistema de que se trate. El conjunto de elementos que referenciamos con un código puede crecer o decrecer y, en el primero de los casos indicados, aparecen dos situaciones diferentes: que se añadan nuevos elementos o que alguno de los existentes se descomponga en dos. Todos estos casos se reducen a tres, desde el punto de vista operativo: supresión, adición o inserción de códigos. Respecto de la supresión de códigos, es necesario tener en cuenta, solamente, que la eliminación no afecte al proceso de los códigos que permanecen; pueden surgir problemas, por ejemplo, si suprimido un código (y aparecido, por tanto, un «hueco» en la codificación) el sistema controla la detención o el cambio de proceso cuando se rompa la secuencia de los códigos presentes; otro conflicto surge si la aplicación examina la presencia de algunos elementos clave y el código suprimido corresponde, precisamente, a uno de esos elementos.

La adición de elementos al final de la serie codificada normalmente no crea demasiados problemas; hay que asegurarse de que existen códigos diferentes para todos los elementos presentes y futuros del conjunto a codificar: una típica situación de conflicto en esta línea se presenta cuando se ha previsto un dígito (dos, tres...) para designar los elementos de un conjunto o las situaciones posibles y aparecen, posteriormente, más de diez (cien, mil...) casos posibles. El problema más delicado se presenta con la inserción de códigos. En efecto, si no está prevista esta inserción en la propia estructura del código, la situación es de solución complicada. Para permitir la inserción de elementos, existen varias técnicas, como veremos, pero en el fondo el procedimiento es siempre el mismo: establecimiento de un código con «huecos»; bien porque el código está organizado en series (con códigos disponibles al final de cada serie), bien porque a todo lo largo de la secuencia de códigos existen espacios (un caso típico de este tipo está representado con la numeración de las líneas de un programa, que suelen numerarse de diez en diez o, incluso, de cien en cien al escribir el programa por primera vez, para poder intercalar líneas posteriormente).



Elección de un código

Para la adecuada elección de un código deben seguirse una serie de pasos de definición del problema:

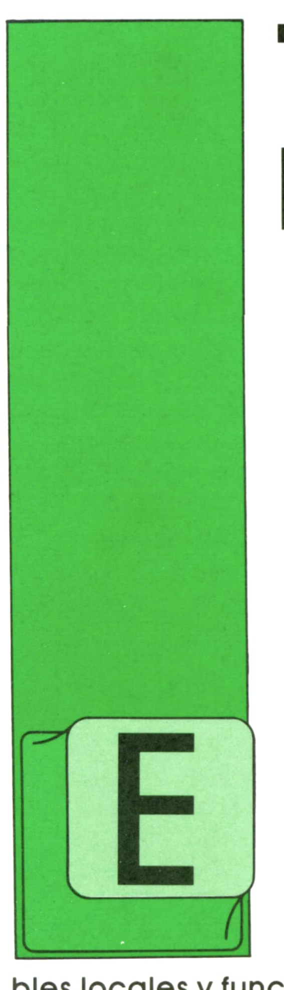
— Ante todo, examinar la (o las) utilidad(es) que va a tener el código; para ello conviene hacer una relación de todas las aplicaciones en que va a ser utilizado dicho código, anotando junto a cada una de ellas el tipo de proceso en que va a ser aplicado el código y la característica clave (de entre las reseñadas anteriormente) que deben cumplir, en consecuencia, el código.

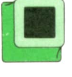
— Estudiar el número de elementos o situaciones de que consta el conjunto a codificar. Es fundamental tener en cuenta no sólo la situación actual, sino la evolución futura previsible, para saber hasta qué punto es importante o no utilizar un código con posibilidad de ampliación o inserción, si se producirán bajas, etc.

— Es útil, asimismo, examinar la distribución de los elementos por grupos y, por tanto, la futura distribución de los códigos, cara a la construcción de series diversas o el establecimiento de una única secuencia de códigos.

— Como en cualquier otra decisión que esté relacionada con la estructura externa de la organización, conviene discutir la codificación que se va a implantar con sus futuros usuarios. Incluso, si es posible, sería conveniente «probar» la codificación antes de su implantación definitiva.

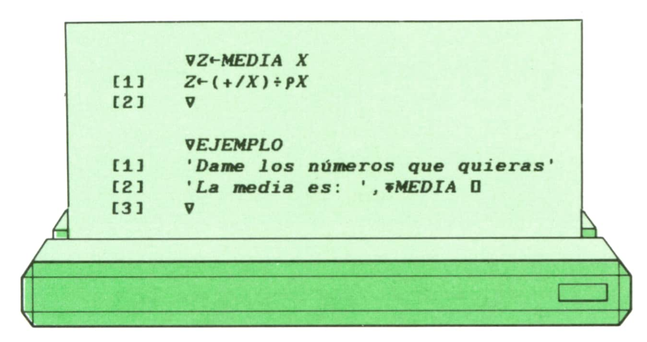
TECNICAS DE PROGRAMACION





Programación modular en el lenguaje APL

EN APL es posible utilizar la programación modular con tanta facilidad y libertad como en PASCAL, pues también disponemos de la posibilidad de definir variables locales y funciones con nombre y resultado. Veamos cómo se programaría en este lenguaje nuestro ejemplo del capítulo anterior:



Observaremos lo siguiente:

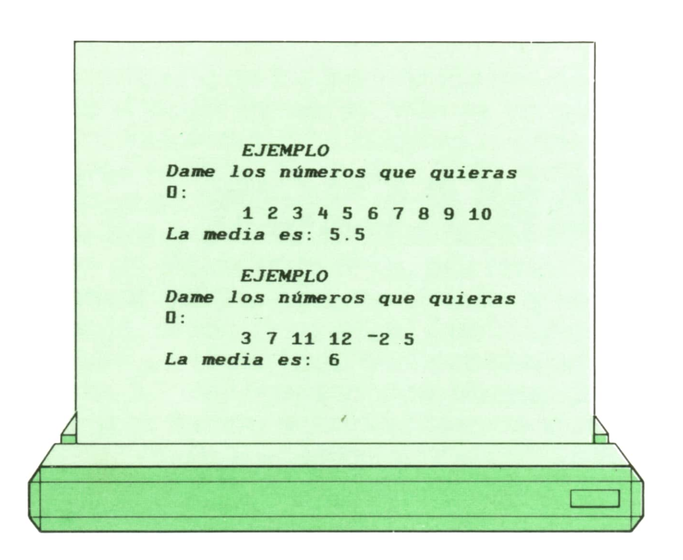
1. En APL se utiliza el triángulo invertido (signo «nabla») para entrar en el modo de definición de un programa, subrutina o función. Las líneas de las funciones se numeran automáticamente de manera consecutiva, aunque es posible insertar líneas si se desea. La función MEDIA recibe un argumento o parámetro (X), cuyo tipo no es preciso definir, y calcula un resultado (Z). Tanto X como Z son variables locales, que no entran en conflicto con otros usos externos de los mismos nombres. Como en PASCAL, tan sólo se considera global (es decir, accesible desde el exterior) el nombre de la propia función (MEDIA, en nuestro caso).

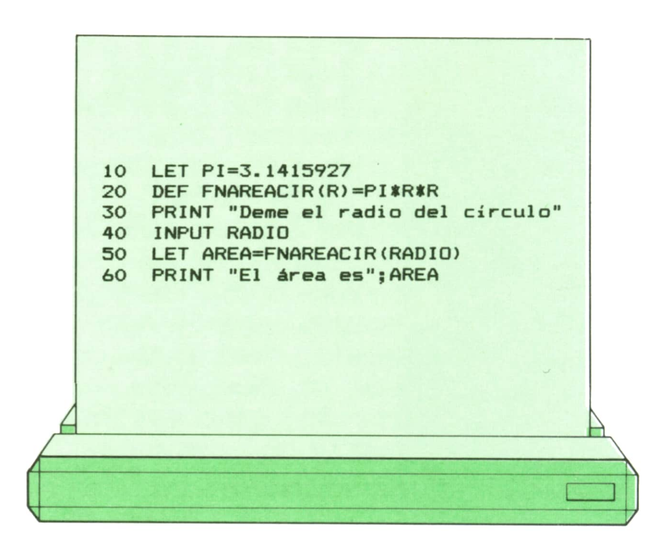
2. La función tiene una sola línea ejecutable y calcula la media sumando todos los elementos de X (lo que se consigue en APL con la expresión +/X) y dividiendo por su número (la letra griega «rho», aplicada a una variable, nos devuelve el número de elementos de ésta).

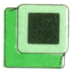
3. El programa principal se define igual que la función MEDIA. Su nombre es EJEMPLO, y no tiene resultado ni parámetros, ni variables locales de ningún tipo. Este programa consta de dos líneas. La primera escribe en la pantalla el mensaje «Dame los números que quieras», pues en APL no es necesario utilizar palabras reservadas (como PRINT o WRITELN) para escribir un texto o resultado sobre la pantalla. Basta con escribir el texto entrecomillado o la operación cuyo resultado queremos conocer.

La segunda línea pide los valores de esos números (pues el cuadrado, en APL, significa que se desean obtener valores del teclado, todos los que se den hasta presionar la tecla ENTER). A continuación, la misma línea calcula la media de dichos números y genera el mensaje final: «La media es:...». El símbolo representado por una T pequeña con un circulito en el palo vertical convierte el valor numérico de la media a la cadena de caracteres equivalente, para poderlo concatenar con el mensaje encerrado entre comillas.

Por último, veamos cómo se ejecuta el programa anterior:







Procedimientos y funciones

En general, en casi todos los lenguajes de programación se distinguen dos clases de subrutinas o módulos, que reciben el nombre de «funciones» y de «procedimientos». A veces, su tratamiento es bastante diferente.

Una «función» es un subprograma que posee un resultado explícito, con el que se puede operar. Las funciones se invocan utilizando su nombre dentro de una expresión. En este capítulo y el anterior hemos visto un ejemplo de una función PASCAL y de una función APL. En ambos casos servían para calcular la media. En el lenguaje BASIC, sin embargo, el uso de las funciones está bastante restringido, y tan sólo se pueden utilizar las que el intérprete o compilador conozca previamente o algunas otras, definidas por el programador, que tienen muchas restricciones. La más importante es que la definición de la función debe ser una expresión, por lo que no es posible construir funciones con más de una línea. Además, para definir una función no se pueden utilizar bloques secuenciales de instrucciones, ni sentencias de bucle, ni instrucciones condicionales, ni transferencias de ningún tipo. Por estas razones, el uso de las funciones BASIC no es muy grande.

Para definir una función BASIC se utiliza la sentencia DEF FN. Veamos un ejemplo:

Obsérvese que también en BASIC se puede dar una lista de uno o varios argumentos o parámetros, escritos entre paréntesis detrás del nombre de la función, que siempre debe comenzar por las letras FN. Estos argumentos son locales, es decir, no interfieren con otros valores que puedan tener las variables del mismo nombre situadas fuera de la función. Además, la definición de la función debe preceder a cualquier uso que se haga de la misma.

Veamos un ejemplo de la ejecución del programa anterior:



Por el contrario, un «procedimiento» es un subprograma que no tiene resultado explícito, por lo que no se puede operar con él. Los procedimientos pueden constar siempre de una o más instrucciones, que se ejecutan sucesivamente de una forma totalmente idéntica a las del programa principal. Se trata, pues, de secciones separadas de éste, que se agrupan porque van a ser ejecutadas varias veces, invocándolas desde distintos lugares del programa principal o de otros subprogramas, o simplemente para facilitar la legibilidad de nuestros programas o porque algún día podremos hacer uso

de estos módulos en entornos de programación diferentes.

Todos los ejemplos de subprogramas BASIC que vimos en el capítulo anterior eran procedimientos. Como vimos allí, los procedimientos BASIC se invocan con la instrucción GOSUB y no permiten la posibilidad de definir variables locales o pasar parámetros.

En PASCAL, un procedimiento se escribe exactamente igual que un programa principal ordinario, sólo que en lugar de utilizar la palabra PROGRAM seguida de un nombre, se usa la palabra reservada PROCEDURE, que puede ir seguida de una lista de argumentos entre paréntesis, exactamente igual que en el caso de las funciones. El resto del procedimiento es semejante a un programa principal: hay una zona declarativa, donde pueden declararse los tipos de las variables utilizadas en el procedimiento, y una zona ejecutable, donde se colocan las instrucciones del procedimiento formando un bloque BEGIN-END.

Las variables declaradas dentro de un procedimiento se consideran locales a éste, y sus valores no entran en conflicto con los que pueden tener las variables del mismo nombre situadas fuera del procedimiento. Estas últimas son inaccesibles dentro del procedimiento. En cambio, las variables que hayan sido declaradas antes del procedimiento, en el programa que engloba a éste (el principal u otro módulo o subrutina cualquiera) tales que su nombre no coincida con el de una variable local, se consideran variables globales y pueden utilizarse dentro del procedimiento sin definición previa.

Un procedimiento PASCAL se invoca desde el programa principal o desde otro módulo escribiendo simplemente su nombre.

Veamos una versión del ejemplo del programa PASCAL del capítulo anterior que utiliza procedimientos.

```
Program EJEMPLO;
TYPE serie = array [1..10] of real;
Function MEDIA (N:integer; X:serie):real;
     {Subrutina que calcula la media
      de N números}
   VAR i:integer; total:real;
   BEGIN
     total:=0;
     for i:=1 to N do total:=total+X[i];
     MEDIA:=total/N
   END;
VAR datos:serie;
PROCEDURE Leedatos(n:integer);
   VAR i:integer;
   BEGIN
     {Este procedimiento lee n números}
     writeln ('Escriba ',n,' números');
     for i:=1 to n do readln(datos[i])
   END;
 PROCEDURE Escribevalor(v:real);
   BEGIN
     writeln ('El resultado es: ',v)
   END;
 BEGIN
    {Este programa lee 10 números
     y calcula su media}
    Leedatos(10);
    Escribevalor(MEDIA(10,datos))
 END.
```

Observemos en este programa las siguientes peculiaridades:

1. Dentro del programa principal, y en su sección declarativa, definimos dos procedimientos y una función.

2. La primera declaración del programa principal es la del tipo «serie», que se encuentra antes que la declaración de la función y de los dos procedimientos. Por tanto, esta variable es global y pueden utilizarla tanto el programa principal, como la función MEDIA, como los dos procedimientos.

3. La segunda declaración del programa es la función MEDIA, en cuya descripción no vamos a entrar, pues es idéntica a la del capítulo anterior.

4. La tercera declaración del programa EJEMPLO es la variable «datos», de tipo «serie». Por la posición que ocupa la declaración, esta variable no será accesible desde la función MEDIA (que no la necesita), pero sí desde los dos procedimientos.

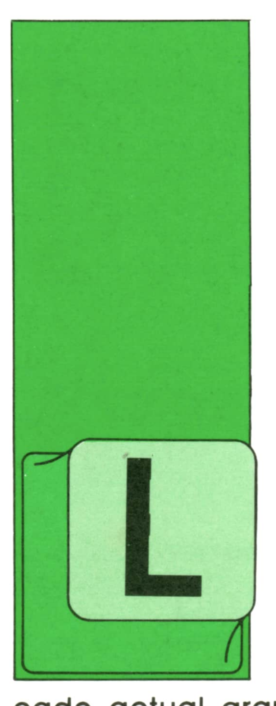
5. A continuación tenemos la declaración del procedimiento LEEDATOS, que tiene un parámetro, una variable local y una variable global (datos, definida previamente en el programa principal). El procedimiento escribe un mensaje en la pantalla pidiendo el número de datos indicado por *n*, lee los valores y los coloca en la variable datos.

6. Inmediatamente después se declara el segundo procedimiento, ESCRIBEVALOR, que recibe un argumento real y que no hace otra cosa que escribirlo por una pantalla con un mensaje adecuado.

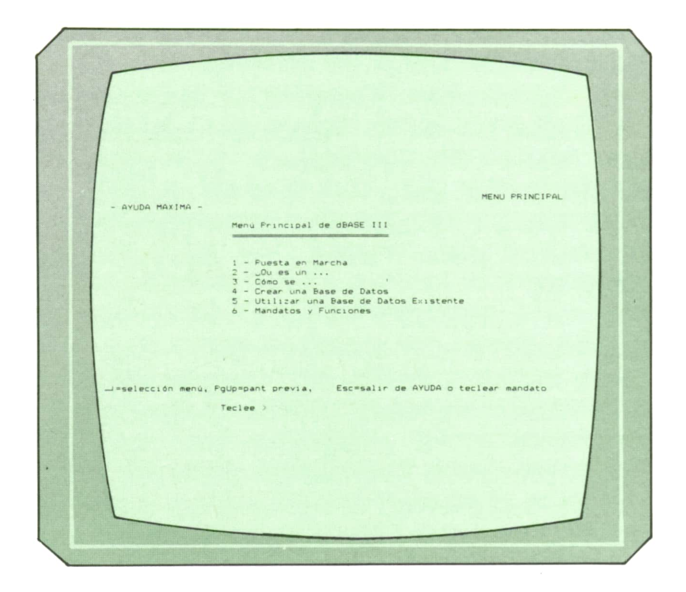
7. Por último, aparece en nuestro ejemplo la parte ejecutable del programa principal, que se limita a pedir los datos, calcular la media y escribir el resultado.

APLICACIONES

SBASE III

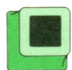


A gestión de los grandes volúmenes de datos constituye un problema que con frecuencia ha de resolver el usuario de un ordenador personal. Existen en el mercado actual gran cantidad de programas que ayudan a la gestión de la información. Uno de ellos es el dBASE, programa que constituye un gestor de bases de datos tipo relacional.



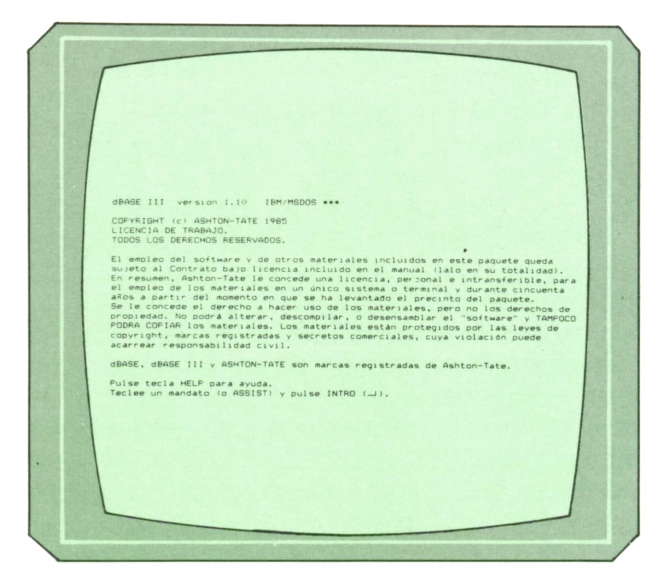
Este programa servirá tanto para llevar una simple agenda telefónica como para gestionar complejas estructuras de ficheros y realizar programas de todo tipo en un tiempo muy corto.

Por ello, incluye gran cantidad de funciones y de comandos.



Ficheros DBASE III

En principio un fichero consta de un cierto número de registros; éstos pueden ser de longitud fija o variable, los campos de longitud variable o número, que permiten cambiar cualquier número de registro.



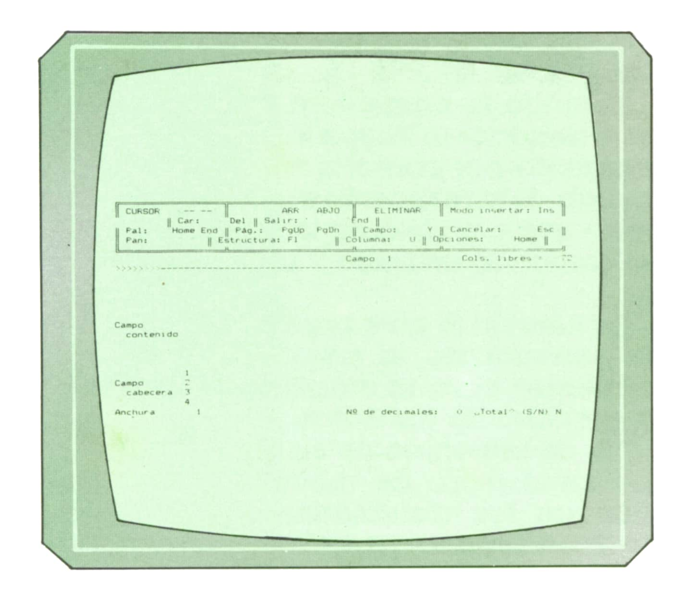
DBASE III permite la importación de ficheros de otros programas, así como la conversión de ficheros de datos a ficheros de tipo texto que pueden ser leídos sin problemas por gran cantidad de procesadores de textos y hojas de cálculo.

Los campos pueden ser numéricos, alfanuméricos, de fecha, lógicos y memo.

Los numéricos y alfanuméricos son del mismo tipo que los habituales de otras aplicaciones y lenguajes.

Los campos de tipo fecha almacenan fechas en el formato estándar MM/DD/AA, que es el que se utiliza en Estados Unidos.

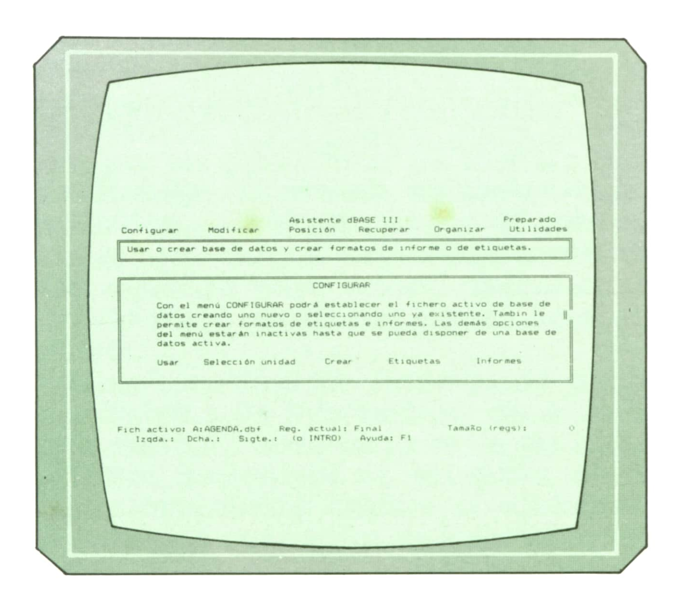
Los campos lógicos abarcan un valor verdadero (TRUE) o falso (FALSE).



El número máximo de campos en un registro es de 128, y aunque el número de registros es ilimitado, el de la longitud máxima es de 510 K, aproximadamente.

El acceso a los ficheros puede hacerse tanto de forma secuencial como directa; en este último método existen dos formas:

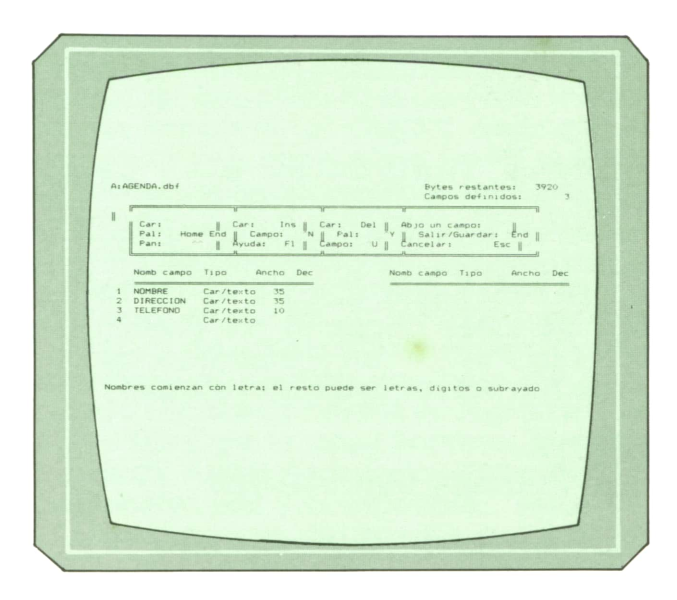
Uno, utilizando el fichero sin hacer ninguna operación extra, y la segunda, indexando el fichero, es decir, creando un fichero de índices, con lo que el acceso de datos se reduce considerablemente, empleando un máximo de dos segundos para ficheros grandes.



En los ficheros pueden añadirse o insertarse ficheros, si bien esta última posibilidad resulta algo lenta cuando los ficheros son grandes, ya que debe proporcionar un espacio en el fichero para el registro nuevo a base de mover una parte de éste.

A la hora de consultar los datos del dBASE III, ofrece la posibilidad de crear formatos de informes de una manera sencilla, quedando abandonados estos formatos por uno posterior.

La salida de los datos puede hacerse indistintamente por la impresora, la pantalla, e incluso las dos a la vez.



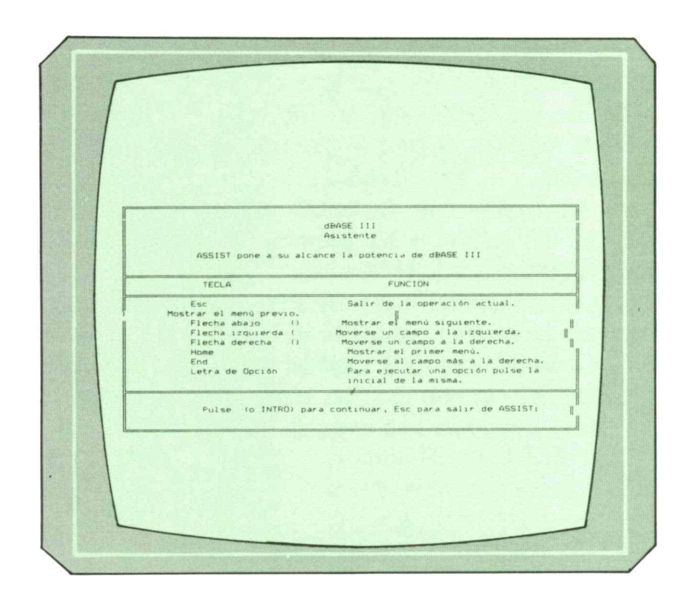
El programa proporciona a la vez multitud de comandos interactivos para la confección de los formatos de pantalla por el propio usuario.

El programa incluye también otras opciones sobre los ficheros como son el borrado de registros, la eliminación de registros borrados, la recuperación de éstos, la ordenación de ficheros por uno o varios campos, etc.

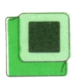
Asimismo el programa contiene comandos interactivos con los que se puede realizar consultas al fichero, cálculos matemáticos y gestionar el sistema y el entorno del ordenador.

Claro está que todas estas opciones no sirven de gran ayuda si no se pudiesen almacenar para su uso posterior.

Pues bien, dBASE III ofrece al usuario un completísimo lenguaje de programación en el que se pueden utilizar todos los comandos normales del dBASE, incluyendo



órdenes, sentencias, etc., como WHILE...WEND, IF...THEN...ELSE, CASE..., que hacen de este lenguaje una muy potente herramienta del desarrollo del programa. En conclusión, dBASE III es uno de los más potentes paquetes de gestión de datos que existen en el mercado, que incluye numerosas ayudas al usuario.



Paquete autoasistido

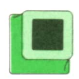
La mayoría de los comandos y funciones necesarias para empezar a trabajar con esta base de datos están asistidas de un programa de ayuda que guían al usuario en la realización de ficheros y en su manipulación. Junto a los dos discos que componen el sistema se facilita un tercero de ejemplos y enseñanza del paquete.

El comando ASSIST abre paso a una serie de menús, ayuda a través de la cual el usuario puede ir creando ficheros e introduciendo modificaciones dentro de él.

Cuando se comete un error de sintaxis, el ordenador lo indica inmediatamente y pregunta por pantalla si se desea recibir ayuda. Si la respuesta es afirmativa, aparece una pantalla en la que se describe el comando utilizado y su sintaxis correcta.

La función FL sirve para poner en la parte superior de la pantalla los diversos controles para la modificación, borrado y escritura de los datos.

La documentación suministrada por el paquete está, de momento, en inglés, aunque los distribuidores de ASHTHON TATE en España ya están preparando la versión castellana. Es completa y fácil de manejar. El manual se divide en varias secciones. La primera de ellas, tutorial, introduce al usuario del paquete en la mayoría de los principales comandos del sistema. La segunda parte, de referencia, describe el paquete, la estructura de la base y el tipo de ficheros de datos. La tercera sección es un índice más completo y detallado de todos los comandos y funciones. Junto a ellos se suministra un pequeño paquete recordatorio.

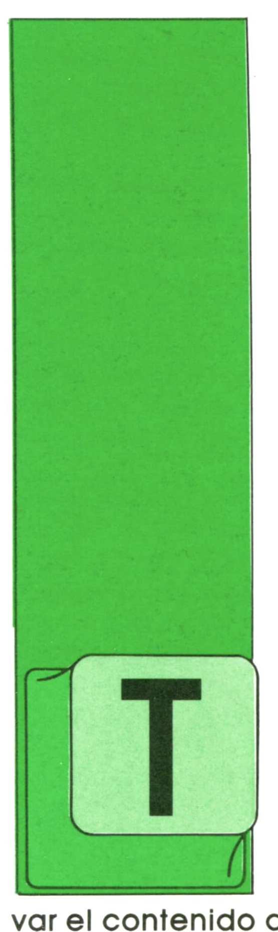


Menú principal de dBASE III

1. Puesta en marcha
2. ¿Qué es un...
3. Cómo se...
4. Crear una Base de Datos
5. Utilizar una Base de Datos existente
6. Mandatos y funciones.

⏎ = selección de menú, PgUp = pant previa, Esc = Salir de AYUDA o teclear mandato Teclee >

PASCAL





Ficheros

TODOS los programas que hemos realizado hasta el momento han guardado sus datos en la memoria del ordenador; por todos es sabido que la única forma de preservar el contenido de ésta es, en principio, mantener el ordenador conectado (salvo en algunos ordenadores portátiles que permiten la desconexión sin que se borre la memoria) y sin dedicarlo a ninguna otra tarea que utilice la memoria para distintas cosas.

Está claro que los ordenadores no habrían alcanzado la enorme difusión de nuestros tiempos si no existiera algún sistema para guardar los datos que se han introducido, o los nuevos datos obtenidos por los programas, de manera que estén disponibles para su uso en cualquier otro momento, aunque el ordenador se haya estado dedicando a otra tarea o incluso haya estado desconectado. De estas cuestiones vamos a hablar ahora.

Existen multitud de sistemas de almacenamiento de información; de ellos el más versátil y extendido entre los ordenadores personales es, sin duda, el de disco magnético flexible o «diskette». Aunque los conceptos que vamos a exponer se van a tratar en plan general, se hará, no obstante, con la mirada puesta en los discos flexibles.

Desafortunadamente, al ser estas cuestiones tan dependientes de cada máquina en concreto, el PASCAL estándar es poco explícito sobre ellas, por lo que hay prácticamente tantas maneras de programarlas como versiones de compilador se han desarrollado.

Por ello, se va a explicar lo que hay al respecto en el PASCAL estándar, aunque a la hora de confeccionar ejemplos utilizaremos una versión concreta de compilador. Para esto hemos escogido el TURBO PASCAL de la casa Borland, que, entre otros, funciona con los ordenadores personales IBM y compatibles; éstos son los ordenadores personales dotados de disco flexible más difundidos en la actualidad y el compilador es, sin duda, el que más ha contribuido a la difusión del PASCAL y una de la herramientas de desarrollo más vendidas en la historia de los ordenadores personales.

Ficheros secuenciales

Se denomina fichero o archivo («file» en inglés) a una secuencia de datos del mismo tipo de los que sólo uno de ellos está disponible en un momento dado. Imaginemos una cinta de magnetófono en la que se han grabado palabras una detrás de otra; el conjunto de palabras es lo que se denominaría un «fichero de palabras» y, como es lógico, en un momento dado sólo es posible escuchar una de ellas. Los ficheros de oficina son también un buen ejemplo.

Los datos que se guardan en los dispositivos de almacenamiento externo de un ordenador adoptan también esta estructura, de manera similar a como se en-

cuentran las palabras guardadas en la cinta del ejemplo. Los datos se encuentran uno detrás de otro en algún tipo de soporte físico (una cinta o un disco magnético, usualmente), de manera que, en un momento dado, sólo uno de ellos se puede leer o grabar.

Al igual que en la cinta podría haber diferentes conjuntos de palabras (por ejemplo, la lista de los reyes Godos, los elementos de la tabla periódica, etc.), en un dispositivo de almacenamiento puede haber diferentes ficheros, a cada uno de los cuales se le habrá dado un nombre distinto.

Cuando la única forma de llegar a un elemento específico de un fichero es comenzar por el primero e ir recorriendo un elemento tras otro hasta llegar al deseado, se dice que el fichero es SECUENCIAL (o de acceso secuencial). Es el caso de las cintas: la única forma segura de encontrar la palabra buscada es ponerse al principio y escuchar una tras otra hasta llegar a ella.

Para cambiar o añadir información a un fichero secuencial, se hace igual que como se haría para sustituir o añadir palabras a la cinta: se va recorriendo el fichero elemento a elemento hasta llegar a la posición deseada y entonces se graba la información.

Hay una serie de programas ya preparados en los ordenadores que se encargan de manejar los dispositivos de almacenamiento; son parte de lo que se conoce como «sistema operativo». Cuando se está trabajando con ficheros, estos programas se encargan de leer y grabar los datos en ellos cuando nuestro programa PASCAL se lo pide, quedando a su cargo el manejo de una serie de punteros propios que les indican en qué elemento de un fichero nos encontramos en un momento dado, y en qué zona concreta del medio físico (cinta, disco..) se encuentra cada fichero. En principio, no es necesario saber más sobre ellos.

En PASCAL es posible definir variables de tipo fichero secuencial, con la particularidad de que los datos que albergan no están en la memoria del ordenador, sino que corresponden a los datos de un fichero almacenado en el dispositivo externo, y de que el número de datos no está definido *a priori*. Lo único que se tiene en memoria en un momento dado es la copia del elemento del fichero en que nos encontramos; a su vez, es posible modificar o añadir un elemento en esa posición con datos procedentes de la memoria.

También se puede, por supuesto, avanzar al siguiente elemento o posicionarse en el primero de todos, así como inaugurar ficheros o hacerlos desaparecer.

Aunque teóricamente sería posible tener ficheros con todos sus componentes almacenados en memoria, pocos compiladores lo permiten y, además, sólo serían útiles en casos muy especiales de proceso de datos.

El tipo de variable se describe poniendo las palabras reservadas FILE y OF seguidas del tipo de elemento que constituye el fichero, que se puede declarar previamente o describirlo sobre la marcha. Para un fichero que contuviera los nombres de los reyes Godos:

```
type
  ReyGodo_t = array [1..15] of char;

var
  (* "fichero de datos tipo ReyGodo_t": *)
  F        : file of ReyGodo_t;

  Tabla : array [1..33] of ReyGodo_t;
  I        : integer;
  Rey     : ReyGodo_t;
```

Para que la variable F se pueda utilizar, hace falta indicar primero al PASCAL cómo se llama el fichero en que se encuentran sus datos y en qué dispositivo se encuentra, caso de existir más de uno; todo ello es parte del proceso denominado «apertura» del fichero. No hay normas sobre cómo hacer esto, por lo que depende de cada compilador.

Supongamos que esta operación ya se ha realizado; hemos dicho que en todo momento, y de manera automática, se tiene en memoria una copia del elemento del fichero sobre el que nos encontramos. Esa copia sería en el ejemplo una variable de tipo ReyGodo_t y, como no tiene ningún identificador asociado, para referirse a ella se escribe F^, que viene a significar algo como «la copia del elemento del fichero F en que nos encontramos ahora». Esta variable puede utilizarse como cualquier otra del mismo tipo, y su contenido puede modificarse

con vistas a ser transferido después al fichero. Por ejemplo:

```
Rey := F^,
F^ := 'Teudiselo          ';
```

Para operar con ficheros se tienen las siguientes funciones y procedimientos predefinidos:

EOF (F). Esta función (End Of File, fin de fichero) devuelve el valor lógico TRUE cuando nos hemos pasado de largo y nos hemos posicionado más allá del último elemento del fichero asociado a F.

RESET (F). Este procedimiento hace que nos coloquemos al principio del fichero con vistas a su lectura. Tras su ejecución, F^ contendría el primer elemento (a no ser que el fichero estuviera vacío, en cuyo caso su contenido sería imprevisible y EOF (F) devolvería TRUE).

GET (F). Este otro, sin embargo, hace que se avance al elemento siguiente a aquél en que nos encontrábamos, pasando F^ a tener una copia suya. Por tanto, si no hubiera elemento siguiente, EOF (F) pasaría a valor TRUE y F^ tendría un contenido indefinido.

Con ellos, si, por ejemplo, quisiéramos leer del fichero todos los nombres de reyes que contuviese guardándolos en la variable Tabla para su posterior utilización, podríamos hacer:

```
I:=1;                        (* Empezar por el primer elemento       *)
reset (F);                   (* Leer el fichero desde el principio   *)
while not eof(F) do          (* Mientras estemos sobre un elemento:  *)
  begin
   Tabla [I]:= F^;           (* Guardar elemento en la tabla         *)
   get (F);                  (* Pasar al siguiente                   *)
   I:=I+1                    (* Incrementar índice                   *)
  end;
```

Se ha utilizado una estructura WHILE, pues si el fichero estuviera vacío, no habría que guardar ni un dato en Tabla.

Imaginemos ahora que tenemos las notas de un examen guardadas en un fichero. Si definimos la variable Notas como FILE OF REAL, para calcular la nota media podríamos hacer:

```
Numero := 0;
Suma      := 0.0;
reset   (Notas);
while not eof (Notas)do
   begin
    Suma:= Suma + Notas^; (*añadir la nueva nota *)
    get (Notas);
    Numero:= Numero + 1  (* incrementar el contador de notas *)
   end;
if Numero = 0  then writeln ('Fichero vacío.')
               else writeln ('Nota media= ',Suma/Numero:6:2);
```

Para modificar o incorporar nueva información a un fichero se dispone de los siguientes procedimientos estándar:

REWRITE (F). Este procedimiento hace que el fichero asociado a F se vacíe, perdiéndose sus datos y quedando preparado para empezar a guardar nuevos datos en él desde su comienzo. Tras su ejecución, EOF (F) pasa a valer TRUE.

PUT (F). Al ejecutarse PUT (F), el contenido de F^ se transfiere al fichero, justo en la posición en que nos encontrábamos, pasándose a continuación a la siguiente posición. Por tanto, si antes de ejecutarse estuviéramos más allá del último elemento, el nuevo quedaría a continuación, siendo ahora él el último, y quedando nuevamente posicionados más allá de éste.

Con ellas, para guardar los elementos de Tabla en el fichero asociado a F haríamos:

```
rewrite (F);
for I:= 1 to 33 do
   begin
      F^ := Tabla (1);
      put (F)
   end;
```

Para añadir nuevas notas al fichero Notas, justo a continuación de las ya existentes, haríamos:

```
(* Nos ponemos al principio y avanzamos hasta llegar al final: *)
reset (Notas);
while not eof(Notas) do get(Notas);

writeln ('Introduzca una nota negativa para acabar.');
repeat
  LeerNota (S);   (* procedimiento para leer notas desde teclado *)
  if S >= 0 then
    begin
     Notas^ := S;
     put (Notas)
    end
until S < 0;
```

LeerNota sería un procedimiento escrito por nosotros y S una variable de tipo REAL.

En el caso de los reyes Godos, como el número de elementos está predefinido, podríamos utilizar un fichero de elementos del mismo tipo que la tabla, con lo que, al haber sólo uno, la forma de proceder sería mucho más sencilla (y rápida):

```
program ReyesGodos;

type
 ReyGodo_t = array [1..15] of char;
 Tabla_t   = array [1..33] of ReyGodo_t;
 Fichero_t = file of Tabla_t;

var
 F     : Fichero_t;
 Tabla : Tabla_t;

begin
 (* ...apertura del fichero... *)

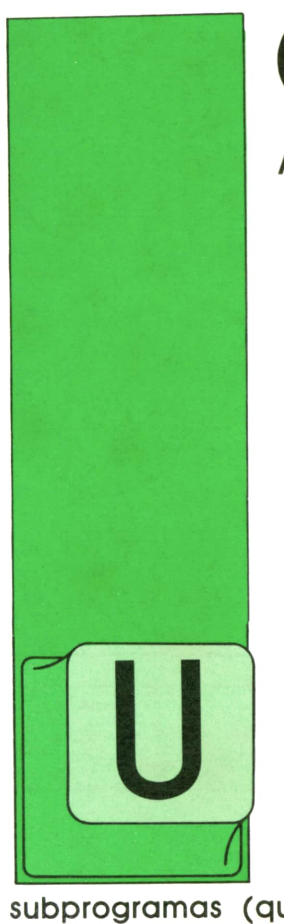
 (* ...lectura de la tabla:    *)
 reset (F);
 if not eof(F) then Tabla := F^
              else writeln ('Fichero vacío.');

 (* ... *)
end.
```

Se ha definido previamente el tipo de fichero a modo de ejemplo exclusivamente.

OTROS LENGUAJES

ADA-2





Partes del lenguaje

UN programa ADA se compone de una o más unidades de programa, las cuales pueden ser compiladas separadamente. Las unidades de programas pueden ser subprogramas (que definen algoritmos ejecutables), paquetes (los cuales definen conjuntos de entidades), o tareas (las cuales definen procesos concurrentes, o corrutinas). Cada unidad, normalmente, está formada por dos partes:

— Una especificación, que contiene la información que debe ser accesible por otras unidades (equivalente al módulo de definición del MODULA-2).

— El cuerpo, que contiene los detalles de la implementación, que son totalmente transparentes al resto de las unidades (equivalente al módulo de implementación del MODULA-2).

Esto permite la realización modular del software, lo que favorece el diseño y mantenimiento de programas muy grandes.

Un programa ADA, normalmente, utiliza unidades de programa que se encuentran en la biblioteca de utilidades, lo que dota de una gran consistencia, ya que casi todas las operaciones más usadas, como las de entrada salida, son definidas como cualquier otra unidad, de forma que para cada ordenador concreto sólo varía el cuerpo de implementación, permaneciendo igual el bloque de definición. Esto favorece la portabilidad del software de un ordenador a otro completamente diferente.

Describamos ahora brevemente las partes del bloque de definición.

Los subprogramas son las unidades básicas en la definición de algoritmos. Estos pueden ser de dos tipos, procedimientos y funciones.

Un paquete es la unidad básica para definir un grupo de entidades relacionadas lógicamente.

Una tarea es la unidad básica para definir una secuencia de acciones, que pueden ser ejecutadas, en paralelo, con otras unidades similares.

El cuerpo de una unidad de programas consta, normalmente, de dos partes:

— La declaración, en la cual se definen las entidades lógicas que se van a utilizar en la unidad (variables, tipos de datos, funciones, procedimientos, etc.).

— Una secuencia de sentencias que definen el proceso de ejecución.



Tipos predefinidos de datos

Cada objeto en el lenguaje pertenece a un tipo que puede tomar una serie de valores y al que se le pueden aplicar un conjunto de operaciones.

Existen cuatro clases de tipos:

— Escalares: Enumerados (o subrango) y numéricos. Los predefinidos son BOOLEAN y CHARACTER, como enumerados, e INTEGER y DURATION como numéricos.

— Compuestos: Cuya estructura está formada por elementos de otros tipos. Pueden ser matrices o registros.

— Tipos de acceso: Permiten la construcción de estructuras de datos creadas por la ejecución de localizadores. Permitiendo asignar a un mismo objeto tipos e identificadores diferentes, según nos interese, pudiendo alterarse éstos en tiempo de ejecución. Este tipo innovador permite una gran flexibilidad en el montaje final de diferentes bloques, eliminando los posibles problemas de incompatibilidad de tipos, uno de los graves problemas del lenguaje MODULA-2.

— Tipo privado: Son tipos ocultos de uso interno de un paquete cuyas características se hacen visibles, en parte, a los demás paquetes, ocultando los detalles menos importantes de la implementación concreta.

También existe el concepto de subtipo (similar al subrango de otro tipo en PASCAL) con limitación de operadores y rango.

A pesar de la gran profundidad sintáctica del lenguaje ADA, el número de palabras reservadas es pequeño (ver figura siguiente).

abort	declare	generic	of	select
accept	delay	goto	or	separate
access	delta		others	subtype
all	digits	if	out	
and	do	in		task
array		is	package	terminate
at			pragma	then
	else		private	type
	elseif	limited	procedure	
	end	loop		
begin	entry		raise	use
body	exception		range	
	exit	mod	record	when
			rem	while
		new	renames	with
case	for	not	return	
constant	function	null	reverse	xor

Otra característica nueva implementada para favorecer la portabilidad del software en diferentes máquinas es la traducción sintáctica. Esta se basa en la traducción de una cadena de caracteres (STRING) pertenecientes a un conjunto (SET) diferente (normalmente proviene de un dispositivo no compatible de otro ordenador), traduciendo unívocamente los códigos de los diferentes caracteres mediante una tabla de traducción.

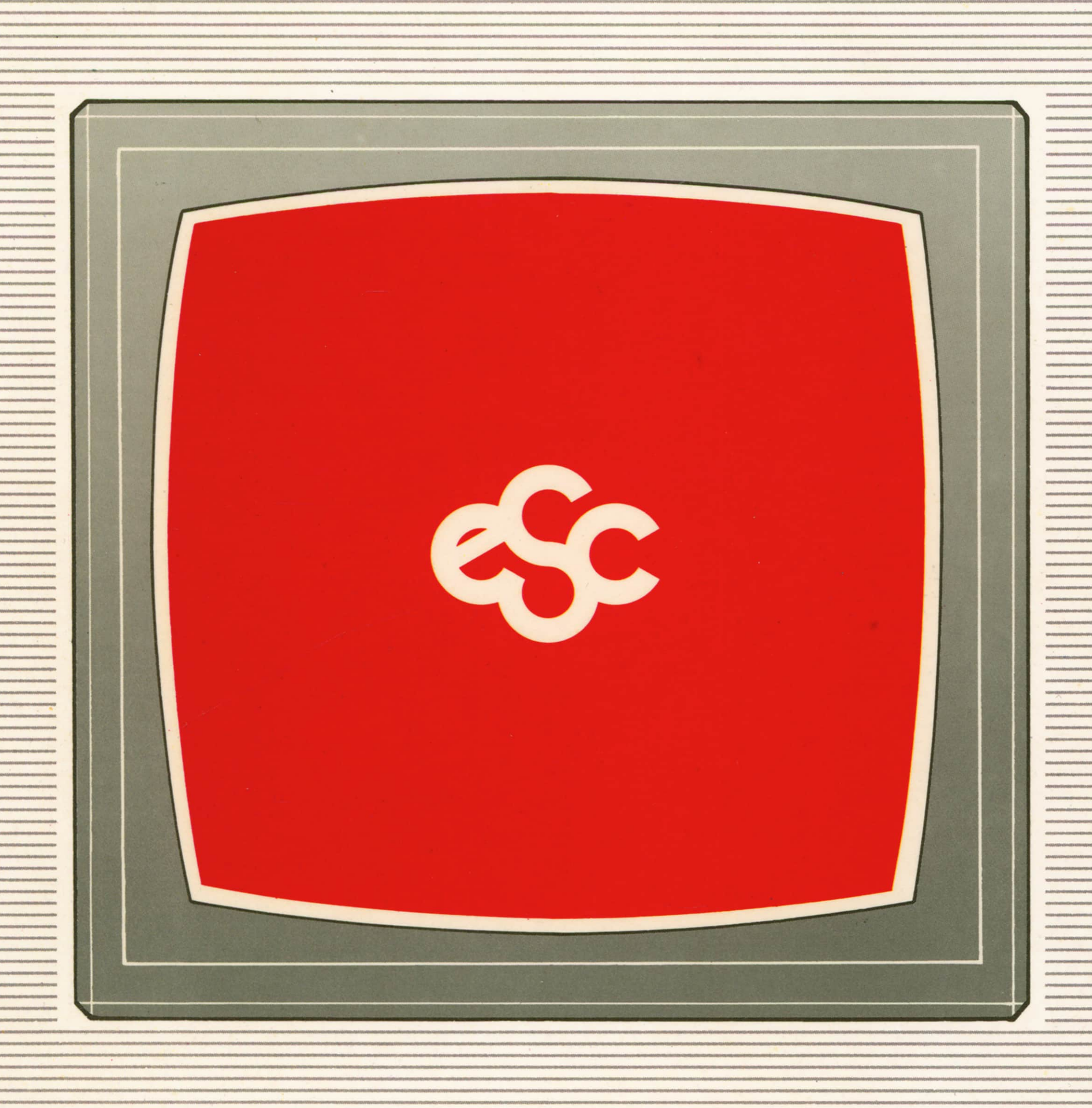
De cualquier forma, el lenguaje ADA utiliza preferentemente el código ASCII.

La sintaxis de las declaraciones, asignaciones y definiciones de tipo son muy similares a las del PASCAL; diferenciándose en ligeros matices léxicos.

Las cadenas de caracteres son llamadas SLICES (en vez de STRINGS, como suele ser habitual).

Por supuesto, las declaraciones permiten partes variantes, para una mayor flexibilidad en la definición, dependiendo de parámetros.

En el próximo capítulo detallaremos la sintaxis del lenguaje.



▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼